Learning Python

第4版
涵盖Python 2.6和3.x

Python

学习手册



O'REILLY®
机械工业出版社 China Machine Press
华章科技

Mark Lutz 著
李军 刘红伟 等译

O'Reilly□□□□□□

Phthon□□□□

——□4□

Learning Python,Fourth Edition

[□]□□□□Lutz□M.□　□

□□　□□□　□□

ISBN□978-7-111-32653-3

□□□□□□□□□□□□□□2011□□□□□□□□□□□□□□

□ □

O'Reilly Media,Inc.□□

□□□

□□

□□□4□

□□Python 3.0□Python 2.6

□□□

□□□□□□□

Python 2.6□Python 3.0□□□□□□□

Python 3.0□□□□□□□

□□□□

□□□□

□□□□□□□□□□

□□□□□□□□

□□□□

□□□□□□□□

□□□□□□□

□□

□□□□

□□

□□□□ □□□□
□1□ □□□□
□□□□□□Python
□□□□
□□□□
Python□“□□□□”□
□□□Python□□□□□□□
□□□□□□Python
□□Python□□□□□□
□□□□
□□□□□□
Internet□□
□□□□
□□□□□
□□□□
□□□□□□□□□□□□
□□□□□□□□□□□□XML□□□□□
Python□□□□□□
Python□□□□□□□□□
□□□□
□□

□□□
□□□□
□□□
□□□□
□□□□
Python□□□□□□□□□□□□
□□□□
□□□□
□□□□
□2□ Python□□□□□□
Python□□□□□
□□□□
□□□□□□
Python□□□
□□□□□□□
Python□□□□□□
□□□□□□
□□□□□□□
□□□□□□
□□□□□□
□□□□

□□□□

□□□□

□3□　□□□□□□

□□□□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□

□□□□□□□□

□□□□□

□□□□□□□□□□

□□□□□□□□□

UNIX□□□□□□(#!)

□□□□□□□

□Windows□□□□□

input□□□

□□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□

import□reload□□□□□□□

□□exec□□□□□□

IDLE□□□□

IDLE□□
□□IDLE
□□IDLE□□
□□□IDE
□□□□□□
□□□□□
□□□□□□□□□□□
□□□□□□□□□□□
□□□□□□□□
□□□□□
□□□□□
□□□□□
□□□□□
□□□□□ □□□
□□□□□ □□□□□
□4□ □□Python□□□□
□□□□□□□□□□
Python□□□□□□□
□□
□□□

□□□□□

□□□□

□□□□□□□

□□□□

□□□□□□□□□□

□□□□

□□

□□□□

□□□□□□□

□□□□

□□

□□□□

□□□□

□□□□□for□□

□□□□□

□□□□□□if□□

□□

□□□□□□□

□□

□□□□□□□□
□□□□□□□
□□□□□□□□□□□□
□□□□□□□
□□□□□□
□□□□□
□□□□□
□□□□□
□5□ □□
Python□□□□□□
□□□□□
□□□□□□□□□□
Python□□□□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□
□□□□□□□□□□□□
□□□□□□□□□□Floor□□□□□□□
□□□□□
□□
□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□

□□□□□□□

□□□□

□□□□

□□

□□□

□□□□

□□□□

□□□□

□□□□

□6□　□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□□□□

□□□□□□□□

□□□□

□□□□□□□□□□

□□□□□□□□

□□□□□□□□□

□□□□

□□□□
□□□□
第7□ □□□
□□□□□
□□□□□□□□□□□□□
□□□□□□□□□□□□□
raw□□□□□□□
□□□□□□□□□□□□□
□□□□□□□□□□
□□□□
□□□□□
□□□□□□□
□□□□□
□□□□□□□□□□□□□□
□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□
□□□□□□□□□□□□Python 3.0□□□□
□□□□□□□□□□
□□□□□□□□□□□□□□
□□□□□□□□□□□□

□□□□□□□□□□□

□□□□

□□□□□□□□□□

□□□□□□□□

□%□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□

□□□□

□□□□

□8□ □□□□□

□□

□□□□□□□□

□□□□□□□

□□□□□□□□

□□□□□□□□

□□□□□□□

□□

□□□□□□□□

□□□□□□□□
□□□□□□□
□□□□□□□
□□□
□□□□□□□□□
□□□□□□□□□□
Python 3.0□□□□□□
□□□□
□□□□
□□□□
□9□ □□□□□□□□□
□□
□□□□□□□□□
□□□□□□□□□□□□
□□
□□□□
□□□□
□□□□□□□□□
□□□□□□□
□□□□□□
□□□□□□

□□VS□□
□□□□□□□□□□
Python 3.0□□□□□
Python□□□□□□
Python□□□□□
Type□□
Python□□□□□
□□□□□
□□□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□
□□□□□□□□□□□□□□
□□□□
□□□□
□□□□
□□□□ □□□
□□□□ □□□□□
□10□ Python□□□□
□□Python□□□□
Python□□□
□□if□□□

Python□□□□□
Python□□□□□
□□□□□□□□□□
□□□□□□
□□□□□□□□□□
□□□□□□□□□□□□
□□□□□□□□□□□□□□
□□□□□□□□□□□□□□
□try□□□□□□
□□□□□□
□□□□
□□□□
□□□□
□11□ □□□□□□□□□□
□□□□
□□□□□□□
□□□□
Python 3.0□□□□□□□□
□□□□□□□
□□□□□□
□□□□□□

□□□□□
□□□□□□□□□□□□□□□
□□□□□
Python 3.0□print□□
Python 2.6 print□□
□□□□□□□
□□□□□□□□
□□□□□
□□□□□
□□□□□
□12□ if□□□□□□□
if□□
□□□□□
□□□□□
□□□□□
Python□□□□
□□□□□□□
□□□□□□□
□□□□□□□
□□□□□
if/else□□□□□

□□□□

□□□□

□□□□

□13□　while□for□□

while□□

□□□□

□□

break□continue□pass□□□else

□□□□□□

pass

continue

break

□□else

for□□

□□□□

□□

□□□□□□

□□□□□□while□range

□□□□□□range□□□

□□□□□range

□□□□□zip□map

□□□□□□□□□enumerate
□□□□
□□□□
□□□□
□14□ □□□□□□□□□□□□□
□□□□□□□
□□□□□
□□□□□iter□next
□□□□□□□□□
□□□□□□□
□□□□□□□□□
□□□□□□□□□□□
□□□□□□□□□
□□□□□□□
Python 3.0□□□□□□□□□
range□□□
map□zip□filter□□□
□□□□□VS□□□□□
□□□□□□□
□□□□□□□
□□□□

□□□□

□□□□

□15□　□□

Python□□□□

#□□

dir□□

□□□□□□__doc__

PyDoc□help□□

PyDoc□HTML□□

□□□□□

□□□□

□□□□□□

□□□□□□□□□□

□□□□

□□□□

□□□□

□□□□　□□□

□□□□　□□

□16□　□□□□

□□□□□□

□□□□

def□□
def□□□□□□□□
□□□□□□□□□□□□
□□
□□
Python□□□□
□□□□□□□□□□□□□□
□□
□□
□□□□
□□□□
□□□□
□□□□
□17□ □□□
Python□□□□□
□□□□□
□□□□□□LEGB□□
□□□□□
□□□□□
global□□

□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□□□
□□□□□□□□□
□□□□□□□□□
□□□□□□□□
nonlocal□□
nonlocal□□
nonlocal□□
□□□□□nonlocal
□□□□
□□□□
□□□□
□18□ □□
□□□□
□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□□
□□□□
□□□□

□□
□□□□□□□□□□□□□□□□
□□□□□□□□
Python 3.0 Keyword-Only□□
min□□
□□
□□□
□□
□□□□□□□□□□□□□set□□
□□Python 3.0 print□□
□□Keyword-Only□□
□□□□
□□□□
□□□□
□19□ □□□□□□□□
□□□□□□
□□□□
□□□□□
□□□□□
□□□□VS□□
□□□□□□
□□□□□□

□□□□□□□□□□□□
□□□□□□□
□□□□□
□□□□□
Python 3.0□□□□□□□
□□□□□□lambda
lambda□□□
□□□□□□lambda
□□□□□□□□Python□□□□□□□□□
□□lambda□□□□□
□□□□□□□□□□□map
□□□□□□□□□□filter□reduce
□□□□□
□□□□□
□□□□□
□20□ □□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□
□□□□□□map
□□□□□□□□□□
□□□□□□□□□
□□□□□□□

□□□□□□□□□□
□□□□□□□yield VS return
□□□□□□□□□□□□□□□□□□□
□□□□□□VS□□□□□□□
□□□□□□□□□□□
□□□□□□□□zip□map
□□□□□□□□□□□□
Python 3.0□□□□□□
□□□□□□□□□□
□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□
□□□□□
□□□□□
□□□□□
□□□□□□□□□
□□□□□
□□□□
□□□□□□□□□□□
□□□□□□□□□
□□return□□□□□
□□□□□□□□□□□

□□□□
□□□□
□□□□
□□□□ □□□
□□□□ □□
□21□ □□□□□□□□
□□□□□□□□
Python□□□□
□□□□□□□□□
□□□□□
□□□□□
import□□□□
1.□□
2.□□□□□□
3.□□
□□□□□□
□□□□□□
□□□□□□□□
sys.path□□
□□□□□□
□□□□□□□□□□

□□□□

□□□□

□□□□

□22□ □□□□□□□□□

□□□□□

□□□□□

import□□

from□□

from*□□

□□□□□□□□

import□from□□□□□

□□□□□□□□□□

import□from□□□□

from□□□□□□□

□□□□□□

□□□□□□□□

□□□□□□□□

□□□□□□

□□□□□□□□

□□□□

reload□□

reload□□
□□□□
□□□□
□□□□
第23章　□□□
□□□□□
□□□□□□□□□
__init__.py□□□
□□□□□
□□□□from□□□import□□
□□□□□□□□□
□□□□□□□
□□□□□
Python 3.0□□□□
□□□□□□□□
□□□□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□□
□□□□
□□□□

□□□□

□24□　□□□□□□

□□□□□□□□

□□□from*□□□□_X□__all__

□□□□□□□□□

□□□□□□□□__name__□__main__

□__name__□□□□□

□□□□__name__□□□□□

□□□□□□□□

Import□□□from□□□as□□

□□□□□□□□□

□□□□□□□□□□

□□□□□□□

□□□□□

□□□□□□□□□□□□□□□

from□□□□□□□□□□□

from*□□□□□□□□

reload□□□□from□□

reload□from□□□□□□□

□□□□□from□□□□□□

□□□□
□□□□
□□□□
□□□□ □□□
□□□□ □□OOP
□25□ OOP□□□□□
□□□□□
□□OOP
□□□□□□
□□□□
□□□□□
□□□□
OOP□□□□□□□
□□□□
□□□□
□□□□
□26□ □□□□□□□
□□□□□□□□□
□□□□□□□□□
□□□□□□□□□□
□□□□□

□□□□□□□□□□
□□□□□
□□□□□□□□□□
□□□□□Python□□□
□□□□□
□□□□□□□□□□□□□
□□□□□□□Python□
□□□□□□□□
□□□□
□□□□
□□□□
□27□ □□□□
□□1□□□□□
□□□□□□
□□□□□□
□□□□□□□□□□
□□2□□□□□□□
□□□□
□□3□□□□□□
□□□□□□
□□4□□□□□□□□□

□□□□
□□□□□□□□□□
□□□□□□□□□
□□□□□
□□□□□□□□
OOP□□□□
□□5□□□□□□□
OOP□□□□□□□□
□□□□□□□□
□□6□□□□□□□
□□□□□
□□□□□□□□
□□□□□□□□□
□□□□□□□□
□□□□□□
□□7□□□□□□□□□□□□□□□□□□□□
Pickle□Shelve
□shelve□□□□□□□
□□□□□shelve
□□shelve□□□□
□□□□

□□□□

□□□□

□□□□

□28□　□□□□□□□□

class□□

□□□□

□□

□□

□□

□□□□□□□□

□□□□□□□□□□□

□□

□□□□□□□

□□□□□□□□□

□□□□□□

□□□□

Python 2.6□Python 3.0□□□□□

□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□

Python□□□□□“□”□□□□□□□□□□

□□□□□□
□□□□□□
□□□□□□□
□□□□□□□
□□□□
□□□□
□□□□
□29□ □□□□□
□□□□
□□□□□□□□□□__init__□__sub__
□□□□□□□□□□
□□□□□□__getitem__□__setitem__
□□□□
□□□□□__getitem__
□□□□□□__iter__□__next__
□□□□□□□□
□□□□□□□□□
□□□□□__contains__□__iter__□__getitem__
□□□□□__getattr__□__setattr__
□□□□□□□□
□□□□□□□□□□□□□□□□

__repr__和__str__□□□□□□□□□□
□□□□□□□□□□□□__radd__和__iadd__
□□□□
Call□□□□__call__
□□□□□□□□□□
□□□__lt__和__gt__□□□□□
Python 2.6的__cmp__□□□□□□Python 3.0□□□□□
□□□□□__bool__和__len__
□□□□□□□__del__
□□□□
□□□□
□□□□
□30□ □□□□
Python的OOP
□□□□□□□□□□□□□□□□□
OOP□□□□“□□□”□□
OOP□□□□“□□□”□□
□□□□□□
OOP□□□□“□□”□□
□□□□□□□□
□□□□□□□□

□□□□□□□□□□□□
□□□□□□□□□□□□□□□
□Python 3.0□□□□□□□□□□□
□□□□□□□□□□□□□□
□□□□□“□□”□
□□□□□□□□
□□□□□□□□□□□□□□
□□□□□□□
□□□□□□□□□□□
□□□□
□□□□
□□□□
□31□ □□□□□□□
□□□□□□□
□□□□□□□□□
□□□□□□□□□
□□□
□□□□□
□□□□□□□
□□□□□□□
□□□□□□□

- slots□□
- □□□
- __getattribute__□□□□
- □□

□□□□□□□□□

- □□□□□□□□□□□
- Python 2.6□Python 3.0□□□□□□
- □□□□□□□□□
- □□□□□□□□□
- □□□□□□□□□□□
- □□□□□□□□□

□□□□□□□□□□□□□

- □□□□□□□□
- □□□□□□
- □□□□□□□□
- □□□□□□□

□□□

- □□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□
- □□□□□□□□□□□□□

Python□□□□□□□□□__getattr__□□□□□

“□□□□”

□□□□

□□□□

□□□□

□□□□ □□□

□□□□ □□□□□□

□32□ □□□□

□□□□□□□□

□□□□□□

□□□□□□□□□□

□□□□□□□□

□□□□

□□□□

□□□□□□□□

□□□□

□□□□

□□□□

□33□ □□□□□□□

try/except/else□□

try□□□□

try/else□□

□□□□□□□

□□□□□□□□□□

try/finally□□

□□□□□try/finally□□□□□□

□□try/except/finally□□

□□try□□□□

□□□□□□finally□except

□□try□□□

raise□□

□□raise□□□□

Python 3.0□□□□raise from

assert□□

□□□□□□□□□□□□□□□□□

with/as□□□□□

□□□□

□□□□□□

□□□□

□□□□

□□□□

第34章 □□□□
□□□□□□□
□□□□□□□□
□□□□□□
□□□□□
□□□□□□□□
□□Exception□
□□□□□□
□□□□□□□
□□□□□□
□□□□□□□
□□□□□□□
□□□□□
□□□□
□□□□
第35章 □□□□□
□□□□□□□
□□□□□□□□□
□□□□□□□□
□□□□□□□

□□□□□□□□

□□□□□□□□raise

□□□□□□□□□□□□

□try□□□□□

□□□□□□□□□

□□sys.exc_info

□□□□□□□□□

□□□□□□□

□□□□□□□□except□□

□□□□□□□□□□□□□□□

□□□□□□□

Python□□□

□□□□□□□□□□

□□□□

□□□□□ □□□

□□□□□ □□□□□

□36□ Unicode□□□□□□

Python 3.0□□□□□□□

□□□□□□□□

□□□□□□□

Python□□□□□□

□□□□□□□□□
Python 3.0□□□□□□□
□□□□□□□
□□
□□Unicode□□□
□□ASCII□□
□□□ASCII□□
□□□□□□ASCII□□
□□Unicode□□□□
□□□□
□Python 2.6□□□Unicode□□□
□□□□□□□□□□□
□□Python 3.0 Bytes□□
□□□□
□□□□
□□bytes□□□□□□□
□□□□□□□
□□Python 3.0□□Python 2.6□bytearray□□
□□□□□□□□□□□□□□
□□□□□□
Python 3.0□□□□□□□□□□

□□□□□□□□□□
□□Unicode□□
□Python 3.0□□□□□□Unicode
□Python 3.0□□□BOM
Python 2.6□□Unicode□□
Python 3.0□□□□□□□□□□□
re□□□□□□
Struct□□□□□□□
pickle□□□□□□□
XML□□□□
□□□□
□□□□
□□□□
□37□ □□□□
□□□□□□□
□□□□□□□□□□□□□□□
□□
□□□□
□□□□□
□□□□□
□□□□□□□□□□

□□□

□□□□

□□□□□

□□□□□

□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

__getattr__□__getattribute__

□□□□

□□□□□

□□□□

__getattr__□__getattribute__□□

□□□□□□□

□□□□□□□□□

□□□□□□□□Manager

□□□□□□□

□□□□□□□□

□□□□□□□□

□□__getattr__□□□

□□__getattribute__□□

□□□□

□□□□

□□□□

□38□ □□□

□□□□□□

□□□□□□□

□□□□□□

□□□□□□□□□

□□□□□□□□□

□□□□

□□□□□

□□□□

□□□□□

□□□□□

□□□□□□□□□

□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□□□

□□□□

□□□□□□□□

□□□□□□□

□□□

□□□□□□
□□□□□□□□□□□□□
□□□□□□□□□□□□□
□□□□□□□□□□□□□
□□□□□□□□□
□□□“□□”□“□□”□□
□□□□□□
□□□□□□
□□__dict__□__slots__
□□□□□□□□
□□□□□□
□□□□
Python□□□□□□
□□□□□□□□□
□□
□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□
□□□□
□□□□
□□□□□VS□□□□
□□□□□□□□□□□

□□□□

□□□□

□□□□

□39□　□□

□□□□□□□□□□□□□□

□□□□□□□

“□□”□□□□□

□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□

□□□Type□□□

Class□□□□

□□□□

□□□□

□□□□

□□□□□□□□

□□□□□□□□

□□□□□□□□

□□□□□□□□□□

□□□□

□□□□□□□□

□□□

□□□□ □□□□

□□□□ □□□□□

□□□□ □□□□□

□□□□ □□

□□□□ □□

□□□□ □□OOP

□□□□ □□□□□

□□□□

□□□□

O'Reilly Media,Inc.□□

□□□

□□□

Python□□□

□□□

□□

□□□□□Python□□□□□□□□□□Python□□□□□□□□□□

□□□4□

□□□4□□□□3□□□□□□□□□□

·□□□Python 3.0□Python 2.6□□□□□□Python 3.0□□□□Python 2.6□□□□□□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·□□

□□2009□□□□□□□□□□□□Python□□□□——Python 3.0□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□Python□□□□□□□□□Python 3.0□□□Python□□□□Python 2□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.1□□□□□□□□□□□□□□□□□□□□□format□□□□□□□□□□□□□□□□□□□□□□□□with□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.1□Python 3.0□□□□□□□□□□□□□Python□□□□□□□□□Python□□□□□□□□□"Python 3.0"□□□□□Python 3.0□□□□□□□□Python 3.X□□□□□□□□□□□□□□

□□□

□□□□□□□□□□□□□□Python 3.0□Python 2.6□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ASCII□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□

Python 2.6□Python 3.0□□□□□□□□

Python 3.0□□□□□□□□

表0-1：Python2.6和Python 3.0中的扩展

扩展	介绍的章
Python 3.0中的print函数	11
Python 3.0中的nonlocal x,y语句	17
Python 2.6和Python 3.0中的str.format方法	7
Python 3.0中的字符串类型: str用于Unicode文本，bytes用于二进制数据	7、36
Python 3.0中的文本和二进制文件的区别	9、36
Python 2.6和Python 3.0中的类装饰器: @private('age')	31、38
Python 3.0中的新的迭代器: range、map、zip	14、20
Python 3.0中的字典视图: D.keys、D.values、D.items	8、14
Python 3.0中的除法运算符: 余数、/和//	5
Python 3.0中的集合常量: {a, b, c}	5
Python 3.0中的集合解析: {x**2 for x in seq}	4、5、14、20
Python 3.0中的字典解析: {x: x**2 for x in seq}	4、8、14、20
Python 2.6和Python 3.0中的二进制位字符串支持: 0b0101、bin(I)	5
Python 2.6和Python 3.0中的分数类型: Fraction(1, 3)	5
Python 3.0中的函数注解: def f(a:99, b:str)->int	19
Python 3.0中的Keyword-only参数: def f(a, *b, c, **d)	18、20
Python 3.0中的扩展的序列分解: a, *b = seq	11、13
Python 3.0中可用的针对包的相对导入语法: from	23
Python 2.6和Python 3.0中可用的环境管理器: with/as	33、35
Python 3.0中的异常语法修改: raise、except/as、superclass	33、34
Python 3.0中的异常链: raise e2 from e1	33
Python 2.6和Python 3.0中的保留字的变化	11
Python 3.0中的新式类的取消	31

表0-1：Python2.6和Python 3.0中的扩展（续）

扩展	介绍的章
Python 2.6和Python 3.0中的特性装饰符: `@property`	37
Python 2.6和Python 3.0中的描述符	31、38
Python 2.6和Python 3.0中的元类	31、39
Python 2.6和Python 3.0中的抽象基类支持	28

Python 3.0□□□□□□□□

□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□0-2□□□□□□□□□□□□□□□

表0-2：影响到本书的Python 3.0中的删除（续）

删除的	替代的	介绍的章
print >> F, x, y	print(x, y, file=F)	11
print x, y,	print(x, y, end=' ')	11
u'ccc'	'ccc'	7、36
'bbb' for byte strings	b'bbb'	7、9、36
raise E,V	raise E(V)	32、33、34
except E, X:	except E as X:	32、33、34
def f((a, b)):	def f(x): (a, b) = x	11、18、20
file.xreadlines	for line in file: (or X=iter(file))	13、14
D.keys()等	list(D.keys())（字典视图）	8、14
map(), range(), etc. as lists	list(map()), list(range())（内置函数）	14
map(None, ...)	zip（或手动代码来补充结果）	13、20
X=D.keys(); X.sort()	sorted(D)（或list(D.keys())）	4、8、14
cmp(x, y)	(x > y) - (x < y)	29
X.__cmp__(y)	__lt__、__gt__、__eq__等	29
X.__nonzero__	X.__bool__	29
X.__hex__, X.__oct__	X._index__	29
排序比较函数	使用key=transform或reverse=True	8
Dictionary <, >, <=, >=	Compare sorted(D.items())（或循环代码）	8、9
types.ListType	list（types只用于非内建名称）	9
__metaclass__ = M	class C(metaclass=M):	28、31、39
__builtin__	builtins（重命名）	17
Tkinter	tkinter（重命名）	18、19、24、29、30
sys.exc_type、exc_value	sys.exc_info()[0], [1]	34、35
function.func_code	function.__code__	19、38
由内置函数运行的__getattr__	在包装类中重定义__X__方法	30、37、38
-t,__tt命令行切换	不一致地使用制表符/空格总是一个错误	10、12
一个函数中的from ... *	只能够出现在一个文件的顶层	22
导入模块，在同一包中	from . import mod，包相关的形式	23

表0-2：影响到本书的Python 3.0中的删除（续）

删除的	替代的	介绍的章
class MyException:	class MyException(Exception):	34
exceptions module	内置作用域，库手册	34
thread、Queue modules	_thread, queue（二者都改名了）	17
anydbm module	dbm（改名了）	27
cPickle module	_pickle（改名了，自动使用）	9
os.popen2/3/4	subprocess.Popen（os.popen保留）	14
基于字符串的异常	基于类的异常（Python 2.6中也是如此）	32、33、34
字符串模块函数	字符串对象方法	7
未绑定方法	函数（通过实例调用静态方法）	30、31
混合类型比较、排序	非数字的混合类型比较是错误	5、9

Python 3.0□□□

□□□□

□□□

□□□□□□□□□□

□□□□□□□Python□□□□□□□□□

□Python Essential Reference□□□□□□□□□□□□O'Reilly□□Python Cookbook□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□3□□Python□□□□□□□□

□□□□□□□□□□□

□□□□□□□Python□□□□□□□□□Python□□□□□□□□□□□

□□□

□□□□

□□□

□□□□□□□

□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□

□□

□□□

□□□□□□□□□□□□□□□□□□□%□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□

O'Reilly Media,Inc.

1005 Gravenstein Highway North

Sebastopol,CA 95472

□□□

□□□□□□□□□□□□□2□□□□□□C□807□□100035□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

http://www.oreilly.com

http://www.oreilly.com.cn

□□

□□□2009□□□□□4□□□□□□□□□“□□□”□□□□□□□□□□□□Python 17□□□□□□□□□12□Python□□□□□□□□□□□□

□□□

□□□□□□□□□□□□□□□OQO□□□□□□□□□□□□□Carl Sagan□□□□□□□□□□□□18□□□□□□□□□□□□□□□

□□□□ □□□□

□1□ □□□□

□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

Python□□□□

□□□□□□□□□Python□□□□□□□□□

□□□□

20□□90□□□

Python□“□□□□”□

Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

□□□Python□□□□□□□

□□□17□□Python□□□12□Python□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□C□C++□□□□□□□□□Python□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□Python

□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□100□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□JPMorgan Chase□UBS□Getco□Citadel□□Python□

·NASA□Los Alamos□Fermilab□JPL□□□Python□□□□□□□□□

·iRobot□□Python□□□□□□□□□□□□□□

·ESRI□□□□□GIS□□□□□□□Python□□□□□□□□□□□

·NSA□□□□□□□□□□□Python□

·IronPort□□□□□□□□□□□□□□□100□□□Python□□□□□□□

·OLPC□□Python□□□□□□□□□□□□

□□□□□□□□Python□□□□□□□□□□□□Python□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□

□□Python□□□□□□

Python□□□□□□

□□□□□□

Python□□□□□□□□□□□□□□□□□□□□GUI□□□Python□□□TKinter□□□□□□□□Tk GUI API□□Python□□□□□□□□□□□□□□□□GUI□Python/Tkinter GUI□□□□□□□□□□□□□□□Windows□X Windows□UNIX□Linux□□□Mac OS□Classic□OS X□□□□□□□□□□□□□□□□□□PMW□□Tkinter□□□□□□□□□□□□□□□□□□□C++□□□□□□wxPython GUI API□□□□Python□□□□□□GUI□

□□PythonCard□Dabo□□□□□□□□□□□□□wxPython□tkinter□□□API□□□□□□□□□□□□□□□□□Python□□□□□□GUI□□□□□□□□□□PyQt□□Qt□□□PyGTK□□GTK□□□□PyWin32□□MFC□□□IronPython□□.NET□□□□□Jython□Java□□□Python□□□□□□□2□□□□□□□□□Swing□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Jython□Python Web□□□□□□□□CGI□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Internet□□

Python□□□□□Internet□□□□□□Python□□□□□□

□□□□

□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□C/C++□□□□□□□□□□□□C/C++□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□Sybase□Oracle□Informix□ODBC□MySQL□PostgreSQL□SQLite□□Python□□□□□□□□Python□□□□SQL□□□□□□□□□□□□□□□□API□□□API□□□□□□□□□□□□□□□□□□□

□□□□

□□Python□□□□□□□Python□C□□□□□□□

□□□□□□□□□□□□

□□□□□□□□NumPy□□□□□□□□□□□

□□□□□□□□□□□□□XML□□□□□

Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·□□□□pygame□□□□□Python□□□□□□□□□□□□

·□□PySerial□□□□Windows□Linux□□□□□□□□□□□□□□□

·□PIL□PyOpenGL□Blender□Maya□□□□□□□□□□□□□□□□

·□PyRo□□□□□□□□□□□□□□

·□xml□□xmlrpclib□□□□□□□□□□□□□□□□XML□□□

·□□□□□□□□□□□□□□□□□□shell□□AI□□□

·□□NLTK□□□□□□□□□□

□□□□□□□□PySol□□□□□□□□□□□□PyPI□□□□□□□□□□□□□□

http://www.python.org□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□Python□□Python□□□□□□□□

Python□□□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□

Python□□□□□□□□□

□□□□□□□□□

□□

Python□□□

□□□

Python□□□□□□□□□□□□ANSI C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□PDA□□□□□□□□□□□□□□□□Python□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□

□□□□

□□□□□□□□□□Python□□□□□□□□□□

Python□□□

□□□

Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□”□□□□□□□□Python□C□□API□□□□Python□□□□□□□□C□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□Python□

□□□□Python□C□□C++□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□Python□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□C++□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

Python□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□Python□□□Perl□Tcl□Java□□□□□□□□□□□□□□□□□□□

·□Ruby□□□□□□□□□□□□□□□Ruby□Java□□□□□OOP□□Python□□□□□□□□□Python□□□□□□□□□□□□OOP□□□□□

·□□SmallTalk□Lisp□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□□

□□□□

□□□

□Python□20□□90□□□

□□□

第2章　Python基础知识

本章介绍了Python的一些基础知识，包括Python的安装、Python的运行方式、Python的基本语法3个方面。通过本章的学习，读者可以对Python有一个初步的认识，为后续的学习打下基础。本章的内容比较简单，但是对于初学者来说，Python的安装和运行方式是一个“拦路虎”，所以本章的内容也是非常重要的。

Python的安装

本节介绍了Python的安装方法，包括Python的下载、Python的安装、Python的环境变量的配置。通过本节的学习，读者可以掌握Python的安装方法，为后续的学习打下基础。

Python的安装非常简单，只需要下载Python的安装包，然后双击安装包，按照提示进行安装即可。Python的安装包可以从Python的官网下载，也可以从一些第三方网站下载。

Python的环境变量的配置是为了让Python的命令可以在任何目录下都可以执行。Python的环境变量的配置方法有两种，一种是在Python的安装过程中自动配置，另一种是手动配置。

Python的语法与C语言的语法和Java的语法有很多相似之处，但是也有很多不同之处。Python的语法比较简单，容易学习。Python的语法主要包括以下几个方面：

Python的语法非常简单，A

·Windows□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□



□ 2-1 □□□□Windows□□□□Python□□□□□Python□□□□□

□□□Python□□□□.py□□□□□□□□□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□

Python□□□

□□□

□□□

□ 2-2 Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□

□□C□C++□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□"build"□"make"□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□(□□□Intel□□□□□□)□□□□□□□□□□Python□□□□□□□□□□□

□□□Python□□□□□□□□□□C□C++□□□□□□□□□□□□□□1□□□□□□□□□

PVM□□□□□□□CPU□□□□□□□□□□□□□□□□□□□□□□□□□□□□□CPU□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□eval□exec□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□Python□□□□□□□□□□□Python□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□Python□□□Java□.NET□□□□□□□□□□□□□□CPython□□□□CPython□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□IronPython□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

CPython□Jython□IronPython□□□□□□□□□□□□Python□□□□□□□□□□□□

JIT□□□

□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□“□□□”Python□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□

□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□

·Stackless Python□□□□□□CPython□□□□□□□□□□□□□C□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□

□□□□□□

□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□Python□□□□□□Python□□□□□□

□□□□

1.□□□Python□□□□

2.□□□□□□□

3.□□□□□□□

4.□□□PVM□

5.□□□□□Python□□□□□□□□□□□□□□

6.CPython□Jython□□IronPython□□□□□?

□□□□

1.Python□□□□□□Python□□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□.py□□□□□□□□□

3.□□□□Python□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□.pyc□□□□□

4.PVM□Python□□□□□□Python□□□□□□□□□□□□□□□□□□

5.Psyco□Shedskin□□forzen binaries□□□□□□□□□□□

6.CPython□Python□□□□□□□□Jython□IronPython□□□Python□□□Java□.NET□□□□□□□□Python□□□□□□□□□□

□3□ □□□□□□

□□□□□

□□□□□□□□□□"python"□□□□□□□□□□Python□□□“%”□□□□□□□□□□

```
C:\misc> cd C:\Python30
C:\Python30> python
Python 3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)] ...
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

□Windows□□□□□shell□□□□□python□□□□□□□□IDLE□□□□□□□□□□□□□□□Python□Start□□□□□□□□□□□□"Python(command line)"□□□□□□□□□□□□□2□□□□2-1□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□shell□□□

□□□□□□□

Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□print□□□□□□□□□□□□□

□□□

□□□□□□□□□□□□

□□□□□□□□□□□

□□□

□□□

□□□□□□□□□

□□□□□□□□

□□10□□□

```
>>> for x in 'spam':
...     print(x)                          <== Press Enter twice here to make this loop run
...
```

□□□

□□□□□□□□

□□□□□□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□□vi□Notepad□IDLE□□□□□□□□□□□□ script1.py□□□□□□□□□□□□Python□□□

```
# A first Python script
import sys                          # Load a library module
print(sys.platform)
print(2 ** 100)                     # Raise 2 to a power
x = 'Spam!'
print(x * 8)                        # String repetition
```

□□□□□□□□□□□□□Python□□□□□□□2□□□2□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□#□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

```
% python script1.py > saveit.txt
```

□□□□□□□□□□□□□□□3□□□□□□□□□saveit.txt□□□□□□□□□□□□□□□□□□□□stream redirection□□□□□□□□□□□□□□□□□□Windows□□□UNIX□□□□□□□□□□□□□□Python□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□shell□□□□□□□□□

□□□□□□Windows□□

```
C:\Python30> python script1.py
win32
126765060022822940149670320537 6
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

□□□□□□□□□PATH□□Python□□□□□□

```
D:\temp> C:\python30\python script1.py
win32
126765060022822940149670320537 6
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

□□□□Windows□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□Windows□□□□Windows□□□□□□□□□□□□□□□□□□□

□□□□□□"python"□□□□□.py□□□□□□□□□□Windows□□□□□□□□□□□□□□□□□

```
D:\temp> script1.py
```

□□□□□□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

□□□

UNIX□□□□□(#!)

□□□Python□Linux□□□□UNIX□□□□□□Python□□□□Python□□□□□□□□□□□□□Shell□□□□□csh□ksh□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□UNIX□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□#!□□□□□□□"hash bang"□□□□□□□□□Python□□□□□□□

·□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□chmod+x brian□□shell□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□shell□□□□□□□□□□□□□□□□□□□□□□□□□□

```
% brian
The Bright Side of Life...
```

□Windows□□□□□□□□□□□□□□□□□□UNIX□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□env□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□UNIX Shell□□□□□□□PATH□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

□Windows□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□ Python□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python script1.py
win32
126765060022822940149670320537 6
```

□□□

input□□□

□□□□□□Windows□□□□□□

□□□□□input□□□□□

·□□□shell□□□□□□□□□□□□□□□□□□□□□□□python spam.py□input.txt□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□input□□□□□□10□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□raw_input()□□□□□□input()□□Python 3.0□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□input□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□input□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□“□□□□”□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python
>>> import script1
win32
126765060022822940149670320537 6
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

□□□

□□□from□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□reload□□□□□□□□□□□□□□□□□□□

□Python 3.0□□□

□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□title□□□□□□□□□□□□□□□□□□□□import□□□□□□□□□□□□□□□□□□□

□□□□title□□□□

```
% python
>>> import threenames                    # Grab the whole module
dead parrot sketch
>>>
>>> threenames.b, threenames.c
('parrot', 'sketch')
>>>
>>> from threenames import a, b, c       # Copy multiple names
>>> b, c
('parrot', 'sketch')
```

□□□

□□□

import与reload的区别详解

在开发过程中，经常会使用import与reload来加载模块，但是两者之间有着很大的区别，如果在IDLE中测试模块，更改模块后，使用reload来重新加载模块，但是reload并不支持级联，比如在模块中又引用了其他模块，reload并不会将其他模块重新加载，而是仍然使用已经加载过的模块。如果要使用reload来加载多个模块，则需要逐个进行reload。

在使用from...import来导入模块时，reload/from...import与reload/import...是一样的，即不支持级联。例如在IDLE Run→Run Module（F5）来执行程序时，实际上是先执行了shell命令reload，因此，在交互式环境下，reload与Run→Run Module（F5）的效果是一样的。

在Python 2.x中，reload是内置的（built-in），而到了Python 3.x，则将其移到了imp模块中（我用的是3.3.2），要使用它，必须先导入imp模块，或使用imp.reload()。

在Python 3.4中，imp模块中的reload函数已经被废弃，推荐使用importlib.reload()[1]。

在Python 3.x中，import与reload的区别更大了，Python对其进行了区分，reload的行为更像是其在Python 2.x中的行为。

[1]□□□□□□□□□□□□21□□□□□□□□□□□□□□□□Python□□□□□sys.path□□□□□□□□□□□□Python□□□□sys□□□□□□□□PYTHONPATH□□□□□□□□□□□□□□□□

□□exec□□□□□□

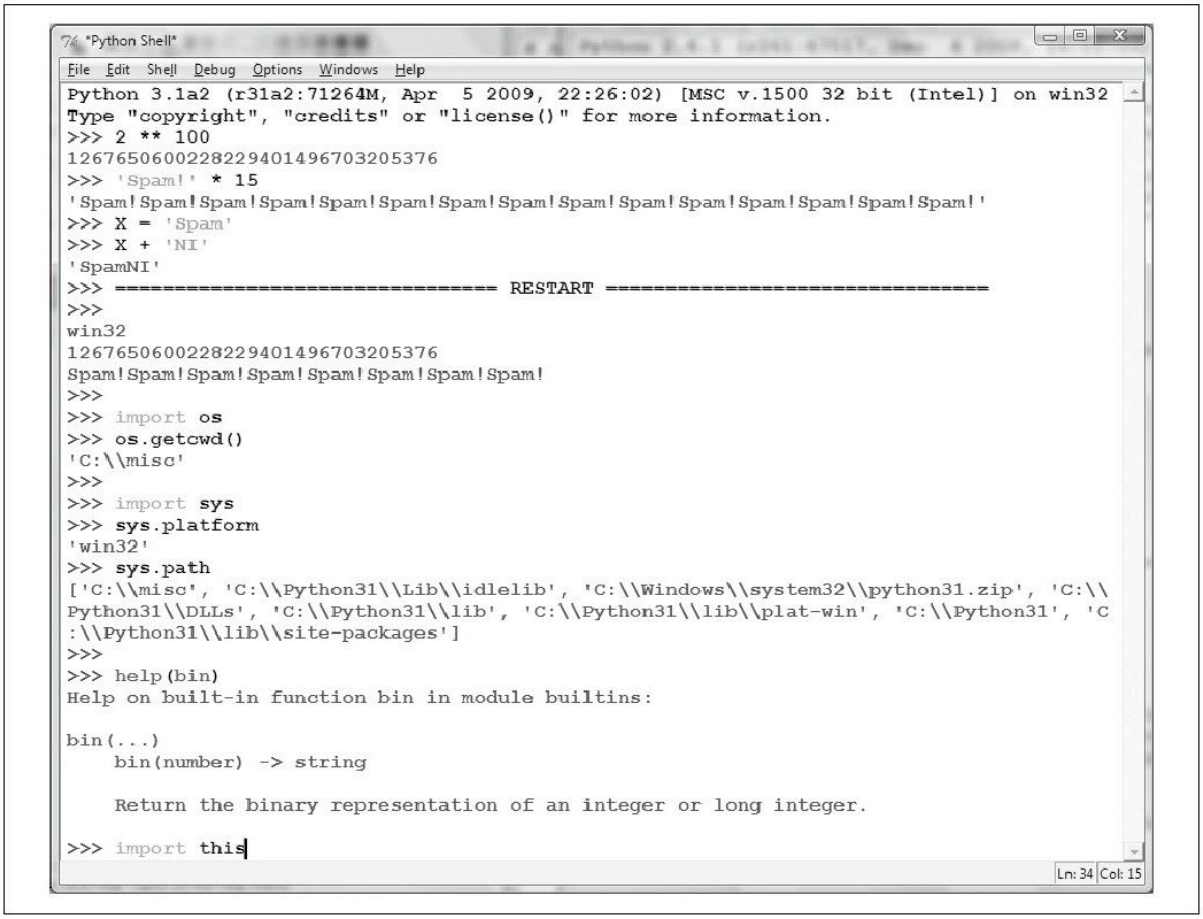
□□□□□□

□□□□□□exec□□□□□□

IDLE□□□□

□□□□

□3-3□□□Windows□□□□□IDLE□□□□Python shell□□□□□



□ 3-3 IDLE□□GUI□□Python shell□□□□Windows□□□□□□File□□□□□□(□□□)□□□□Open...□□□□□□□□□□□□□□□□□□Run□□□□□□□□□□□□Run Module□

□□□□□□□□□□□□IDLE□□□□□□□□□□□□□□□□□□□□Alt-P□□□□□□□□□□□□□□□□□□□Alt-N□□□□□□□Mac□□□□□□□□□Ctrl-P□Ctrl-N□□□□□□□□□□□□□□□□□□□

[1]IDLE□IDE□□□□□□□□□□□□□□□□□□□□□□□□Monty Python□□□Eirc Idle□□□□□

[2]IDLE□Python□□□□□□□□□□Tkinter GUI□□□□□□□□IDLE GUI□□□IDLE□□□□□□□□□□□□□□□□□□□□□□□□Python□□Tkinter□□□□IDLE□Python□Windows□□□□□□IDLE□□□□Linux□UNIX□□□□□□□□□□□□Tkinter□□□□□□yum tkinter□□□□□Linux□□□□□□□□□□□□□□□□□□□□□□□□A□□□□□Mac OS X□□□□□□□□□□□□□□□□□□□□□□□□idle□□□□□□

□□IDLE

IDLE□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

·□□□IDLE□□□□□□□□IDLE□□□□□□□□□□□□□□IDLE□□□□□□Option□□□□Configure□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□IDLE□□□□□□□□□□□□□□□□□□□□□

·□IDLE□□□□□□□□□□□□□□□□□□□

·□□□□IDLE□□□□□□□□□□□□□□□□□□□IDLE□□□□□□□□□□□□□□□□□□□

□□IDLE□□

□□□□□□□□□□□□□□□IDLE□□□□□□□□□□□□□□□□□□□□□point-and-click□□□□□□□□□□□□□IDLE□□□□□□□□Debug□□□□□□□□□□□□□□□□□□□File□□□□□□□□□□□□□□□□□□□

□□□IDE

□□IDLE□□□□□□□□□□□□□Python□□□□□□□□□□□□□IDE□□□□□□□□□□□□□

NetBeans IDE Python□

NetBeans□□□□□□□□□□□□□□GUI□□□Python□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□http://www.python.org□□□□□□□□
Google□□"Python editors"□□□□□□□□□□□□Wiki□□□□□□□□□□□□
Python□□□IDE□□□□□□□□□□□□

□□□□□□

□□□□□

```
#include <Python.h>
...
Py_Initialize();                                    // This is C, not Python
PyRun_SimpleString("x = 'brave ' + 'sir robin'");   // But it runs Python code
```

C□□□□□□□C□□□□□□□□□□□□Python□□□□□□□□Python□□□□□□□□Python□□□□□Python□□□□□□□□□□□□□C□□□□□□□□□□□□Python API□□□□Python□□□□□□□□□□□□□□□□

□□□□□□□□Python/C□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□[1]□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□IDE GUI□□□□□□□□□□□□□□□□□□□□Python □□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□Macintosh□□□□□□□□□□□□□□□Python□□□□□□□Python□□□□□□□□□□□□□□□□□Windows□□□□□□□□□□□□□□□□□□□□…□□□Python□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□Python□□□□□Python□□□□□□os.popen□os.system□□□□□Web□□□□□□□□□□□□□□□Python□□□□□□□Web□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□

□□□□□□□

□□□□□□

·□□print□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□print□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□"I am here"□□□□□□□□□□□□□□□□□□□

□□□

□□□□

□□□□□□□□□□Python□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.□□□Windows□□□□□□□□□□□□□□□□□□□□□

5.□IDLE□□□□□□□□□□□

6.□□2□□□IDLE□□□□□□□□

7.□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□Windows□□□□□□□“□□”□□□□□“□□”□□□"Python"□□□□□"Python(command line)"□□□□□□□□□□□□□□□□□Windows□□□□□□□□□□□Windows□□□□□□□□□□□□□□□python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□IDLE□□□□□□Python shell□□□□□□□□□□□□□□□□□□□

5.□□□□□□□□Python□□□□□□□□

□□□□ □□□

□□□

4.□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□1.5.1□□□□□Python□□□□□□□□□□□□□□□□

□□□□ □□□□□

□4□ □□Python□□□□

□□□□□□□□□□Python□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□“□□”□□“□□”□□□□□□□□□□□□□□□“□□”□□□□□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□□——□□□Python□□□□□□□□□□□□□Python□□□C□□□□□□□□□□□□□□□□□□□

□□3□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□4-1□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□

□□□□□□□4-1□□□□□□□

□□

□□□□□□□□□□□□□□□□□□□□4-1□□□□□□□□□□□□□□□□□□□

□□□

```
>>> import random
>>> random.random()
0.592687352662739 53
>>> random.choice([1, 2, 3, 4])
1
```

Python□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

□□□

```
>>> S
Spam'
>>> S + 'xyz'                    # Concatenation
'Spamxyz'
>>> S                            # S is unchanged
'Spam'
>>> S * 8                        # Repetition
'SpamSpamSpamSpamSpamSpamSpamSpam'
```

□□□□□+□□□□□

□□□□

□□□

□□□□□□□

□□□□□□

```
>>> line = 'aaa,bbb,ccccc,dd'
>>> line.split(',')             # Split on a delimiter into a list of substrings
['aaa', 'bbb', 'ccccc', 'dd']
>>> S = 'spam'
>>> S.upper()                   # Upper- and lowercase conversions
'SPAM'

>>> S.isalpha()                 # Content tests: isalpha, isdigit, etc.
True

>>> line = 'aaa,bbb,ccccc,dd\n'
>>> line = line.rstrip()        # Remove whitespace characters on the right side
>>> line
'aaa,bbb,ccccc,dd'
```

□□□

□□□□

□□□

```
>>> help(S.replace)
Help on built-in function replace:

replace(...)
    S.replace (old, new[, count]) -> str

    Return a copy of S with all occurrences of substring
    old replaced by new. If the optional argument count is
    given, only the first count occurrences are replaced.
```

□□PyDoc□□□

□□□□□□□□□□

□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□Unicode□□□□□□□□□□□□ASCII□□□□□□□□□Unicode□□□□□□
bytes□□□□□□□□□□□□□□□Python 2.6□□Unicode□□□□□□□□□□str□□8□
□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□str□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□

□□

Python□□□

```
>>> L[0]                        # Indexing by position
123

>>> L[:-1]                      # Slicing a list returns a new list
[123, 'spam']

>>> L + [4, 5, 6]               # Concatenation makes a new list too
[123, 'spam', 1.23, 4, 5, 6]

>>> L                           # We're not changing the original list
[123, 'spam', 1.23]
```

□□□□□□□

Python□□□□□□

```
>>> M = ['bb', 'aa', 'cc']
>>> M.sort()
>>> M
['aa', 'bb', 'cc']
>>> M.reverse()
>>> M
['cc', 'bb', 'aa']
```

□□□□□□□□sort□□□□□□□□□□□□□□□□□□□□reverse□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□

□□

Python□□□

□□NumPy□□□□Python□□□□□MatLab□□□□□

□□□□

□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□list comprehension expression□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□if□□□□□□□□□□%□□□□□□□□□□□□□□□□□□□

```
>>> list(map(sum, M))                    # Map sum over items in M
[6, 15, 24]
```

□Python 3.0□□□□□□□□□□□□□□□□□□□□

```
>>> {sum(row) for row in M}              # Create a set of row sums
{24, 6, 15}

>>> {i : sum(M[i]) for i in range(3)}    # Creates key/value table of row sums
{0: 6, 1: 15, 2: 24}
```

□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□

```
>>> [ord(x) for x in 'spaam']            # List of character ordinals
[115, 112, 97, 97, 109]
>>> {ord(x) for x in 'spaam'}            # Sets remove duplicates
{112, 97, 115, 109}
>>> {x: ord(x) for x in 'spaam'}         # Dictionary keys are unique
{'a': 97, 'p': 112, 's': 115, 'm': 109}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

Python□□□□□□□□□□□□□□□□□Monty Python□□□□□□□□□□□□□□□□□□□□□mapping□□□□□□□□□□□□□□□□□□□

□□□

□□□□

□□□

```
>>> rec['name']                          # 'name' is a nested dictionary
{'last': 'Smith', 'first': 'Bob'}

>>> rec['name']['last']                  # Index the nested dictionary
'Smith'

>>> rec['job']                           # 'job' is a nested list
['dev', 'mgr']
>>> rec['job'][-1]                       # Index the nested list
'mgr'

>>> rec['job'].append('janitor')         # Expand Bob's job description in-place
>>> rec
{'age': 40.5, 'job': ['dev', 'mgr', 'janitor'], 'name': {'last': 'Smith',
'first': 'Bob'}}
```

□□□□□□□□□□□□□□□□□□□□job□□□□□□□job□□□□□□□□□□□□□□□□□□□

□□□□□□□Python□□□□□□□

□□□□□for□□

□□□□

```
>>> Ks = list(D.keys())                  # Unordered keys list
>>> Ks                                   # A list in 2.6, "view" in 3.0: use list()
['a', 'c', 'b']

>>> Ks.sort()                            # Sorted keys list
>>> Ks
['a', 'b', 'c']

>>> for key in Ks:                       # Iterate though sorted keys
        print(key, '=>', D[key])         # <== press Enter twice here
a => 1
b => 2
c => 3
```

□□□

for□□□□□□□□□□□□while□□□□□□□□□□□□

□□□□□

□□for□□□□

□□□

□□□□□□if□□

□□□□□

□□□□□□if□□□□□□

□□

□□□□□tuple□□□□"toople"□"tuhple"□□□□□

```
>>> T = ('spam', 3.0, [11, 22, 33])
>>> T[1]
3.0
>>> T[2][1]
22
>>> T.append(4)
AttributeError: 'tuple' object has no attribute 'append'
```

□□□□□□□

□□□□□□

□□

□□□□□Python□□□□□

□□□

□□□□□□□

open□□□□□□□Python□□□□□□□

□□□□□□

□□□□□

```
>>> 1 / 3                                  # Floating-point (use .0 in Python 2.6)
0.3333333333333331
>>> (2/3) + (1/2)
1.1666666666666665

>>> import decimal                         # Decimals: fixed precision
>>> d = decimal.Decimal('3.141')
>>> d + 1
Decimal('4.141')

>>> decimal.getcontext().prec = 2
>>> decimal.Decimal('1.00') / decimal.Decimal('3.00')
Decimal('0.33')

>>> from fractions import Fraction         # Fractions: numerator+denominator
>>> f = Fraction(2, 3)
>>> f + 1
Fraction(5, 3)
>>> f + Fraction(1, 2)
Fraction(7, 6)
```

Python□□□□□□□□□□□□□□□□True□False□□□□□□□□□□□□□□□□

□□□type□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□

□□□□□□□Python□□□□□□□

□□□"self"□□□

□□□□□

□□□□□□□□□□Python□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□□

□□□□

1.□□□None□□□□□□□□□□□□□□□□□□□

6.“□□”□□□□□□□□□□□+□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□5□　□□

□□□□□□□□□□□□Python□□□□□□Python□□□□□□□□□□□□□——□□□Python□□□□□□□□□□□□□□Python□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□

·□□□□

·□□

·□□□□

·□□□□□□□

·□□□□□□□□□□□□

□□□□□□□□□□□□

表 5-1：基本数字常量

数字	常量
1234, -24, 0, 99999999999999	整数（无穷大小）
1.23, 1., 3.14e-10, 4E210, 4.0e+210	浮点数
0177, 0x9ff, 0b101010	Python 2.6中的八进制、十六进制和二进制常量
0o177, 0x9ff, 0b101010	Python 3.0中的八进制、十六进制和二进制常量
3+4j, 3.0+4.0j, 3J	复数常量

□□□□□Python□□□□□

□Python 3.0□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□5-1□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□5-1□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□

□□□□□□□

+□-□*□/□□□□**□□□□

□□□□□□□

pow□abs□round□int□hex□bin□□

□□□□□

random□math□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

Python□□□□□□

□□□□□□

表5-2：Python表达式操作符及程序（续）

操作符	描述
x and y	逻辑与（只有x为真，才会计算y）
not x	逻辑非
x in y, x not in y	成员关系（可迭代对象、集合）
x is y, x is not y	对象实体测试
x < y, x <= y, x > y, x >= y	大小比较，集合子集和超集值相等性操作符
x == y, x != y	
x | y	位或，集合并集
x ^ y	位异或，集合对称差
x & y	位与，集合交集
x << y, x >> y	左移或右移y位
x + y, x - y	加法/合并，减法，集合差集
x * y, x % y, x / y, x // y	乘法/重复，余数/格式化，除法：真除法或floor除法
-x, +x	一元减法，识别
~x	按位求补（取反）
x ** y	幂运算
x[i]	索引（序列、映射及其他）点号取属性运算，函数调用
x[i:j:k]	分片
x(...)	调用（函数、方法、类及其他可调用的）
x.attr	属性引用
(...)	元组，表达式，生成器表达式
[...]	列表，列表解析
{...}	字典、集合、集合和字典解析

□□□□□□□Python 2.6□□□Python 3.0□□□□□□□□5-2□□□□□□□□□□□□□□□□

·□Python 2.6□□□□□□□□□□□□X!=Y□X□□Y□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□X!=Y□□□□

·□Python 2.6□□□□□□□□□□□'X'□repr(X)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□

·在Python 2.X□□□

□□□□□□□□□X+Y*Z□Python□□□□□□□Y*Z□□□□□□□□□X□□□□□“*”□“+”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□A*B□C*D□□□□□□□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□

□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□□□□□□Python□□□□□#□□□□□□□□□□□□□□□□

□Python□□□

[1]□□□□□□□□□□□□□□□□□□□□□#□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□

```
>>> num = 1 / 3.0
>>> num                          # Echoes
0.33333333333333331
>>> print(num)                   # print rounds
0.333333333333

>>> '%e' % num                   # String formatting expression
'3.333333e-001'
>>> '%4.2f' % num                # Alternative floating-point format
'0.33'
>>> '{0:4.2f}'.format(num)       # String formatting method (Python 2.6 and 3.0)
'0.33'
```

□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□str□□□□□□□□□□□□□□□□□□□

```
>>> X < Y < Z          # Chained comparisons: range tests
True
>>> X < Y and Y < Z
True
```

□□false□□□□□□□□□□□□□□□□□□□□□□

```
>>> X < Y > Z
False
>>> X < Y and Y > Z
False

>>> 1 < 2 < 3.0 < 4
True
>>> 1 > 2 > 3.0 > 4
False
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□Floor□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□Python 2.6□□□□□□□□□□□3□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□

X/Y

□□□□□□□□□□□Python 2.6□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

X//Y

Floor□□□□Python 2.2□□□□□□□□□Python2.6□Python3.0□□□□□□□□□□□□□□□□□□□

·□Python 2.6□□/□□□

□□□

□□□Python□math□□□

```
C:\misc> c:\python26\python
>>> 5 / 2, 5 / -2              # Differs in 3.0
(2, -3)

>>> 5 // 2, 5 // -2            # This and the rest are the same in 2.6 and 3.0
(2, -3)

>>> 5 / 2.0, 5 / -2.0
(2.5, -2.5)

>>> 5 // 2.0, 5 // -2.0
(2.0, -3.0)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□math.trunc□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□round□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python
>>> import math
>>> 5 / -2                     # Keep remainder
-2.5
>>> 5 // -2                    # Floor below result
-3
>>> math.trunc(5 / -2)         # Truncate instead of floor
-2

C:\misc> c:\python26\python
>>> import math
>>> 5 / float(-2)              # Remainder in 2.6
-2.5
>>> 5 / -2, 5 // -2            # Floor in 2.6
(-3, -3)
>>> math.trunc(5 / float(-2))  # Truncate in 2.6
-2
```

□□□□□□□□

□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> (5 / 2), (5 / 2.0), (5 / -2.0), (5 / -2)          # 3.0 true division
(2.5, 2.5, -2.5, -2.5)

>>> (5 // 2), (5 // 2.0), (5 // -2.0), (5 // -2)     # 3.0 floor division
(2, 2.0, -3.0, -3)

>>> (9 / 3), (9.0 / 3), (9 // 3), (9 // 3.0)         # Both
(3.0, 3.0, 3, 3.0)
```

□□Python 2.6□□□□□□□□□□□□□□□□

```
>>> (5 / 2), (5 / 2.0), (5 / -2.0), (5 / -2)          # 2.6 classic division
(2, 2.5, -2.5, -3)

>>> (5 // 2), (5 // 2.0), (5 // -2.0), (5 // -2)     # 2.6 floor division (same)
(2, 2.0, -3.0, -3)

>>> (9 / 3), (9.0 / 3), (9 // 3), (9 // 3.0)         # Both
(3, 3.0, 3, 3.0)
```

□□□□□□□□□□□□□□□Python 3.0□/□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□Python□□□□□

Python□□□

□□

□Python□□□

□□□□□□□□□□□□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□□

□□□□□□□□□□□□□□

```
>>> int('64'), int('100', 8), int('40', 16), int('1000000', 2)
(64, 64, 64, 64)

>>> int('0x40', 16), int('0b1000000', 2)          # Literals okay too
(64, 64)
```

□□□□□□□□eval□□□□□□□□□□□□Python□□□□□□□□□□□□□□

```
>>> 0o1, 0o20, 0o377          # New octal format in 2.6 (same as 3.0)
(1, 16, 255)
>>> 01, 020, 0377             # Old octal literals in 2.6 (and earlier)
(1, 16, 255)
```

□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□Python□□□C□□□□□□□□□□□□□□□□□□□

```
>>> X = 0b0001                    # Binary literals
>>> X << 2                        # Shift left
4
>>> bin(X << 2)                   # Binary digits string
'0b100'

>>> bin(X | 0b010)                # Bitwise OR
'0b11'
>>> bin(X & 0b1)                  # Bitwise AND
'0b1'

>>> X = 0xFF                      # Hex literals
>>> bin(X)
'0b11111111'
>>> X ^ 0b10101010                # Bitwise XOR
85
>>> bin(X ^ 0b10101010)
'0b1010101'

>>> int('1010101', 2)             # String to int per base
85
>>> hex(85)                       # Hex digit string
'0x55'
```

□□□□□□□□□□□□□□□“□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□C□□□□□□□□□□□□□□□□□□□

□□□□□□□□□Python 3.1□□□□□bit_length□□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□□Python□□□□□□□□□□□□

```
>>> math.floor(2.567), math.floor(-2.567)          # Floor (next-lower integer)
(2, -3)

>>> math.trunc(2.567), math.trunc(-2.567)          # Truncate (drop decimal digits)
(2, -2)

>>> int(2.567), int(-2.567)                        # Truncate (integer conversion)
(2, -2)

>>> round(2.567), round(2.467), round(2.567, 2)    # Round (Python 3.0 version)
(3, 2, 2.5699999999999998)

>>> '%.1f' % 2.567, '{0:.2f}'.format(2.567)        # Round for display (Chapter 7)
('2.6', '2.57')
```

□□□

```
>>> import math
>>> math.sqrt(144)                  # Module
12.0
>>> 144 ** .5                       # Expression
12.0
>>> pow(144, .5)                    # Built-in
12.0

>>> math.sqrt(1234567890)           # Larger numbers
35136.418286444619
>>> 1234567890 ** .5
35136.418286444619
>>> pow(1234567890, .5)
35136.418286444619
```

□□□□math□□□□□□□□□□□□□□□abs□□□□□□□□□□□□□□□

```
>>> import random
>>> random.random()
0.44694718823781876
>>> random.random()
0.28970426439292829

>>> random.randint(1, 10)
5
>>> random.randint(1, 10)
4

>>> random.choice(['Life of Brian', 'Holy Grail', 'Meaning of Life'])
'Life of Brian'
>>> random.choice(['Life of Brian', 'Holy Grail', 'Meaning of Life'])
'Holy Grail'
```

Random□□□□□□□□□□□□□□□□□□□GUI□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□□□Python□□□□□□□

□□□

decimal□□□

□Python 2.6□Python 3.0□□□

□□□□

Python 2.6□Python 3.0□□□□□□□□□□□□——□□□□□□□□□□□□

```
>>> x + y
Fraction(1, 1)
>>> x - y                                  # Results are exact: numerator, denominator
Fraction(-1, 3)
>>> x * y
Fraction(2, 9)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> Fraction('.25')
Fraction(1, 4)
>>> Fraction('1.25')
Fraction(5, 4)
>>>
>>> Fraction('.25') + Fraction('1.25')
Fraction(3, 2)
```

□□□□

□□□□□□□□□□□□□□□□□□□

```
>>> a = 1 / 3.0                    # Only as accurate as floating-point hardware
>>> b = 4 / 6.0                    # Can lose precision over calculations
>>> a
0.3333333333333331
>>> b
0.6666666666666663

>>> a + b
1.0
>>> a - b
-0.3333333333333331
>>> a * b
0.2222222222222221
```

□□□

```
>>> 1 / 3                                  # Use 3.0 in Python 2.6 for true "/"
0.33333333333333331

>>> Fraction(1, 3)                         # Numeric accuracy
Fraction(1, 3)

>>> import decimal
>>> decimal.getcontext().prec = 2
>>> decimal.Decimal(1) / decimal.Decimal(3)
Decimal('0.33')
```

□□□

```
>>> (2.5).as_integer_ratio()                    # float object method
(5, 2)


>>> f = 2.5
>>> z = Fraction(*f.as_integer_ratio())         # Convert float -> fraction: two args
>>> z                                           # Same as Fraction(5, 2)
Fraction(5, 2)

>>> x                                           # x from prior interaction
Fraction(1, 3)
>>> x + z
Fraction(17, 6)                                 # 5/2 + 1/3 = 15/6 + 2/6

>>> float(x)                                    # Convert fraction -> float
0.33333333333333331
>>> float(z)
2.5
>>> float(x + z)
2.8333333333333335
>>> 17 / 6
2.8333333333333335

>>> Fraction.from_float(1.75)                   # Convert float -> fraction: other way
Fraction(7, 4)
>>> Fraction(*(1.75).as_integer_ratio())
Fraction(7, 4)
```

□□□□□□□□□□□□□□□□□□□□Fraction□□□□□□□□□□□□□□□□□□□

□□□

□□

Python 2.4□□□□□□□□□□——□□□set□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□collection□□□□□□□□□□□□□□□□□□□

□□□

```
>>> z = x.intersection(y)                    # Same as x & y
>>> z
set(['b', 'd'])
>>> z.add('SPAM')                            # Insert one item
>>> z
set(['b', 'd', 'SPAM'])
>>> z.update(set(['X', 'Y']))                # Merge: in-place union
>>> z
set(['Y', 'X', 'b', 'd', 'SPAM'])
>>> z.remove('b')                            # Delete one item
>>> z
set(['Y', 'X', 'd', 'SPAM'])
```

□□□□□□□□□□□□□□□□□□□□len□for□□□□□□□□□□□□□□□□□□□

```
>>> S = set([1, 2, 3])

>>> S | set([3, 4])                    # Expressions require both to be sets
set([1, 2, 3, 4])
>>> S | [3, 4]
TypeError: unsupported operand type(s) for |: 'set' and 'list'

>>> S.union([3, 4])                    # But their methods allow any iterable
set([1, 2, 3, 4])
>>> S.intersection((1, 3, 5))
set([1, 3])
>>> S.issubset(range(-5, 5))
True
```

□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□

```
>>> S1 - {1, 2, 3, 4}                    # Empty sets print differently
set()
>>> type({})                             # Because {} is an empty dictionary
<class 'dict'>

>>> S = set()                            # Initialize an empty set
>>> S.add(1.23)
>>> S
{1.23}
```

□Python 2.6□□□Python 3.0□□□

□□□

```
>>> {x ** 2 for x in [1, 2, 3, 4]}        # 3.0 set comprehension
{16, 1, 4, 9}
```

□□□

□□□

```
{'sue'}

>>> engineers | managers                # All people in either category
{'vic', 'sue', 'tom', 'bob', 'ann'}

>>> engineers - managers                # Engineers who are not managers
{'vic', 'bob', 'ann'}

>>> managers - engineers                # Managers who are not engineers
{'tom'}

>>> engineers > managers                # Are all managers engineers? (superset)
False

>>> {'bob', 'sue'} < engineers          # Are both engineers? (subset)
True

>>> (managers | engineers) > managers   # All people is a superset of managers
True

>>> managers ^ engineers                # Who is in one but not both?
{'vic', 'bob', 'ann', 'tom'}

>>> (managers | engineers) - (managers ^ engineers)        # Intersection!
{'sue'}
```

□Python□□□

□□□

□□Python□□□□□□□□□□□□□bool□□□□□□□□□□□□□□□□□□□□True□False□□□□□□1□0□□□□□□□□□□□□□□□□□□□

```
>>> type(True)
<class 'bool'>
>>> isinstance(True, int)
True
>>> True == 1                    # Same value
True
>>> True is 1                    # But different object: see the next chapter
False
>>> True or False                # Same as: 1 or 0
True
>>> True + 4                     # (Hmmm)
5
```

□□□□□□□□□Python□□□□□□□□□

□□□□

□□□□□Python□□□□□

□□□□

□□□□□Python□□□□□

□□□□

1.Python□□□□2*(3+4)□□□□□□

2.Python□□□□2*3+4□□□□□□

3.Python□□□□2+3*4□□□□□□

4.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

5.□□□1+2.0+3□□□□□□□□□□

6.□□□□□□□□□□□□□□□□□□□□

7.□□□□□□□□□□□□□□□□

8.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

9.□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□14,□2*7□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□10□□6+4□□□□□Python□□□□□□□□□□□□□□□□□□□

8.□□□□oct(I)□hex(I)□□□□□□□□□□□□□□□□□□□□□□□□□□□□%□□□□□□□□□□□□□□□□□□Python 2.6□Python 3.0□□bin(I)□□□□□□□□□□□□□□□□□□□□%□□□□□□□□□□□□□□□□□□□

第6章　□□□□□□

□□□□□□□Python□□□□□□□□□□Python□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□a=3□□□□□□□□□□Python□a□□□□□□□□□□□□□□□□□□□□a□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>> a = 3
```

□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□

1.□□□□□□□□□□□3□

2.□□□□□□a□□□□□□□□□□□□

3.□□□□□□□□□3□□□□

□□□□□□□□6-1□□□□□□□Python□□□□□□□□□6-1□□□□□□□□□□□□□□□□□□□

·□□□

□□□□□□□□□□□□□□

□□□□□□□□□□□□□

□□Python□□□

□□□□□□□

□□□□□□□□□□□□□□□a□□□□□□□□□□□□□□□□□□□□□□□a□□□□□□□□□□□□□□□□□□□

□□□□Python□□□□

□□□□

□□□

```
>>> a = 3
>>> b = a
>>> a = 'spam'
```

□□□□□Python□□□□□

□□□□□□□□□□□□□□□Python□□□a□□□□3□□b□□□a□□□□□□□□□6-2□□□□□□□□□□□□□□□a□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□5□□□□“+”□□□□□□□□□□□□□□□□□b□□□□□□□□□□□□□□□□□□□□□□□□3□□□□□□□□4□□□□□□□□□□□□□□□□□□□

L1□□□□□□□□□2□3□4□□□□□□□□□

□□□□□□□□□copy□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□4□□□7□□□□□□□□□□□□□

```
>>> L1 = [2, 3, 4]
>>> L2 = L1[:]                    # Make a copy of L1
>>> L1[0] = 24

>>> L1
[24, 3, 4]
>>> L2                            # L2 is not changed
[2, 3, 4]
```

□□□□L1□□□□□□□□L2□□□L2□□□□L1□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□

```
>>> L = [1, 2, 3]
>>> M = [1, 2, 3]                    # M and L reference different objects
>>> L == M                           # Same values
True
>>> L is M                           # Different objects
False
```

□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> X = 42
>>> Y = 42                           # Should be two different objects
>>> X == Y
True
>>> X is Y                           # Same object anyhow: caching at work!
True
```

□□□□□□□□X□Y□□□□==□□□□□□□□□□□□□□□is□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□

□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□/□□□□□□□□□□□□□□□□□□□

□□□□

□□□Python□□□□□□□□□Python□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□A□□□□□□□□

```
A = "spam"
B = A
B = "shrubbery"
```

2.□□□□□□□□□□□□□□□□A□□□□

```
A = ["spam"]
B = A
B[0] = "shrubbery"
```

3.□□□□□A□□□□□

```
A = ["spam"]
B = A[:]
B[0] = "shrubbery"
```

□□□□

1.□□A□□□□"spam"□□□□□□B□□□□□□"shrubbery"□□□□□□□□□□□B□□□□□□□□□□□□□□□□□□□□A□B□□□□□□□□□□□□□□□□□□□

第7章 网络编程

网络编程是计算机编程中非常重要的一部分，Python语言——这门在互联网时代诞生的语言，
从一开始就具有强大的网络编程能力。4.1节介绍了Python语言的socket模块，这个模块是网络编程
的基础。

网络编程的基础是socket，socket是计算机网络中两个程序进行双向通信的端点。socket
模块是Python语言对socket的封装，Internet上的Python程序通过socket模块进行通信，
socket模块的函数可以处理Unicode。

Python语言的socket模块是对C语言的socket接口的封装，
Python语言的socket模块的函数与C语言的socket接口的函数一一对应，
Python语言与C语言的socket接口的函数的参数和返回值的类型基本相同。

虽然使用Python语言的socket模块的函数进行网络编程非常简单，
但是网络编程的难点不在于编写程序，而在于理解网络协议——网络
协议规定了网络中的计算机之间如何通信，网络编程的基础是网络协议。

图7-1是网络协议的层次结构图，网络协议分为5层，从上到下依次是应用层、传输层、网络

□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

表 7-1：常见字符串常量和表达式

操作	解释
`s= ''`	空字符串
`s= "spam's"`	双引号和单引号相同
`S = 's\np\ta\xOOm'`	转义序列
`s = """..."""`	三重引号字符串块
`s= r'\temp\spam'`	Raw字符串
`S = b'spam'`	Python 3.0中的字节字符串（参见第36章）
`s = u'spam'`	仅在Python 2.6中使用的Unicode字符串（参见第36章）
`s1 + s2`	合并，重复
`s * 3`	
`s [i]`	索引，分片，求长度
`s [i:j]`	
`len(s)`	
`"a %s parrot" % kind`	字符串格式化表达式
`"a {0} parrot".format(kind)`	Python 2.6和Python 3.0中的字符串格式化方法
`s.find('pa')`	字符串方法调用：搜索
`s.rstrip()`	移除空格
`s.replace('pa', 'xx')`	替换
`s.split(',')`	用展位符分隔
`s.isdigit()`	内容测试
`s.lower()`	短信息转换
`S.endswith('spam')`	结束测试
`'spam'.join(strlist)`	插入分隔符
`S.encode('latin-1')`	Unicode编码等
`for x in S: print(x)`	迭代，成员关系
`'spam' in S`	
`[c * 2 for c in S]`	
`map(ord, S)`	

□□□□□□□□□□□□□□□□□Python□□□□□re□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□□□

□Python□□□□□□□□□□□

```
>>> 'knight\'s', "knight\"s"
("knight's", 'knight"s')
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
>>> len(s)
5
```

□□□□□□□□□□□□□□□□□□□□□□□□ASCII a□□□□□□□□□□□□□□ASCII b□□□□□□□□□□□□□□□□□□□

表7-2：字符串反斜杠字符（续）

转义	意义
\b	倒退
\f	换页
\n	新行（换行）
\r	返回
\t	水平制表符
\v	垂直制表符
\N{id}	Unicode数据库ID
\u*hhhh*	Unicode 16位的十六进制值
\U*hhhhhhhh*	Unicode 32位的十六进制值[a]
\x*hh*	十六进制值
\ooo	八进制值
\0	Null（不是字符串结尾）
other	不转义（保留）

a：\U*hhhh*...转义序列带有八个十六进制数字（*h*）；\u和\U只能使用于Unicode常量之中。

□□□

□□□□□□□□□——□□□□□1□2□□□□□□□□□□□□□□□□□3□□□□□□□□□□

```
>>> s = '\001\002\x03'
>>> s
'\x01\x02\x03'
>>> len(s)
3
```

□□□Python□□□□□□□□□□□□□□□□□

□□□□□7-2□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□“\”□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> x = "C:\py\code"              # Keeps \ literally
>>> x
'C:\\py\\code'
>>> len(x)
10
```

□□□□□□□□7-2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[2]□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“\\”□“\”□□□□□□□□□□□raw□□□□□□□□□□□□□□□□□□□□□□□□□□raw□□□□□

[1]□□□□□□□□□□□□□□□□□□□

raw□□□□□□□

□□□□□□□

```
>>> path = r'C:\new\text.dat'
>>> path                                  # Show as Python code
'C:\\new\\text.dat'
>>> print(path)                           # User-friendly format
C:\new\text.dat
>>> len(path)                             # String length
15
```

□□□

□□□□□□□□□□□□□

□□□raw□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□

□□□□□□□□□

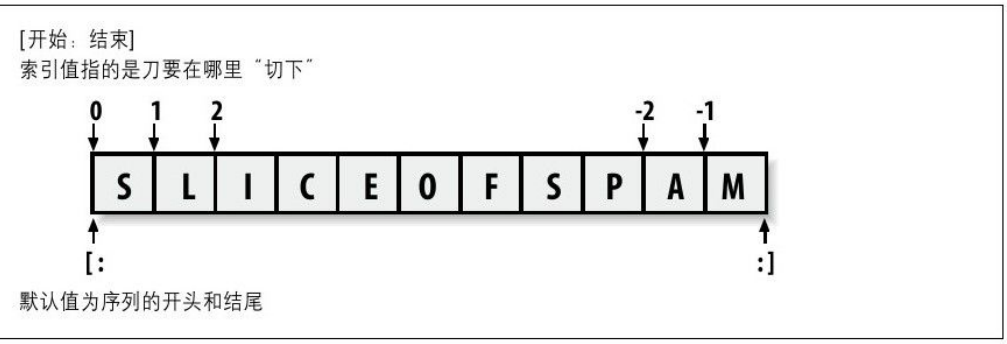
```
>>> print('------- ...more... ---')          # 80 dashes, the hard way
>>> print('-' * 80)                          # 80 dashes, the easy way
```

□□□

□□□

□□□□□

□□□□



□ 7-1 □□□□□□□□□□□□□□□□□□□□0□□□□□□□□□□□□□□□□□□□

□□□

—S[:]□□□□□□0□□□□□□□□□□□□□□□□□□S□□□

□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□□9□□□□□□□□□□□□□□□□□□□

```
>>> S = 'abcdefghijklmnop'
>>> S[1:10:2]
'bdfhj'
>>> S[::2]
'acegikmo'
```

□□□□□□□□□□□□□□□□□□□□□□□"hello"[::-1]□□□□□□□□□"olleh"——□□□□□□□□□□□0□□□□□□□□□□□□□□□□□-1□□

```
>>> S = 'hello'
>>> S[::-1]
'olleh'
```

□□□S[5:1:-1]□□□□□□□□□□2□5□□□□□□□□□□5□4□3□2□□□□□

```
>>> S = 'abcedfg'
>>> S[5:1:-1]
'fdec'
```

□□□□□□□□□□□□□□□□□□□

□□□

```
# File echo.py
import sys
print(sys.argv)

% python echo.py -a -b -c
['echo.py', '-a', '-b', '-c']
```

□□□

□□□□□□□

Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>> print(str('spam'), repr('spam'))
('spam', "'spam'")
```

□□□□□5□□“□□□□□□□”□□□□“str□repr□□□□”□□□□□□□□□□□□□□□□□□□□□int□str□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□+□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> S = "42"
>>> I = 1
>>> S + I
TypeError: cannot concatenate 'str' and 'int' objects

>>> int(S) + I                  # Force addition
43
>>> S + str(I)                  # Force concatenation
'421'
```

□□□

```
>>> str(3.1415), float("1.5")
('3.1415', 1.5)

>>> text = "1.234E-10"
>>> float(text)
1.23400000000000001e-010
```

□□□□□□□□□□□□□□□□□eval□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□

```
>>> S = '5'
>>> S = chr(ord(S) + 1)
>>> S
'6'
>>> S = chr(ord(S) + 1)
>>> S
'7'
```

□□□

□□□□□I□□1□□□□□□2□□□□□□

□□□□□

□□□“□□□□□”□□

```
>>> S = 'spam'
>>> S[0] = "x"
Raises an error!
```

□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

□□□□□

□□□□

□□□

S.isprintable()	S.swapcase()
S.isspace()	S.title()
S.istitle()	S.translate(map)
S.isupper()	S.upper()
S.join(iterable)	S.zfill(width)

□□□Python□□□□□□□□□□□□□□□□□□□

```
>>> S = 'spammy'
>>> S = S.replace('mm', 'xx')
>>> S
'spaxxy'
```

replace□□□

```
>>> where = S.find('SPAM')                  # Search for position
>>> where                                   # Occurs at offset 4
4
>>> S = S[:where] + 'EGGS' + S[(where+4):]
>>> S
'xxxxEGGSxxxxSPAMxxxx'
```

find□□□-1□□□□□□□□□□□□□□□□□□□□□□□□in□□□□□□□□□□□□□□□□□□□□□□□□□□find□□□□□□□□□□□□□□□□□□□

```
>>> S = 'spammy'
>>> L = list(S)
>>> L
['s', 'p', 'a', 'm', 'm', 'y']
```

□□□list□□□——□□□□□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□

```
>>> 'SPAM'.join(['eggs', 'sausage', 'ham', 'toast'])
'eggsSPAMsausageSPAMhamSPAMtoast'
```

□□□

□□□□□□□□□□□□□

□□——□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

```
>>> line
'The knights who say Ni!\n'

>>> line.find('Ni') != -1                    # Search via method call or expression
True
>>> 'Ni' in line
True

>>> sub = 'Ni!\n'
>>> line.endswith(sub)                       # End test via method call or slice
True
>>> line[-len(sub):] == sub
True
```

□□□□□□□□□□□□□□□□□□□□□format□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□Python 3.0□□□□

Python□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□

```
>>> S = 'a+b+c+'
>>> x = S.replace('+', 'spam')
>>> x
'aspambspamcspam'
```

□Python 2.6□□□□□string□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□Python 3.0□□
□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□%□□□

□□□□□□□

1.□%□□□□□□□□□□□□□□□□□□□

```
>>> exclamation = "Ni"
>>> "The knights who say %s!" % exclamation
'The knights who say Ni!'

>>> "%d %s %d you" % (1, 'spam', 4)
'1 spam 4 you'

>>> "%s -- %s -- %s" % (42, 3.14159, [1, 2, 3])
'42 -- 3.14159 -- [1, 2, 3]'
```

□□□□□□□□□□□□□□□□□□□□□□□□'Ni'□□□□□%s□□□□□□□□□□□□□□□□□□□

□□□□□□□printf□□□□□□□

表7-4：字符串格式化代码

代码	意义
s	字符串（或任何对象）
r	s，但使用repr，而不是str
c	字符
d	十进制（整数）
i	整数
u	无号（整数）
o	八进位整数
x	十六进制整数
X	x，但打印大写
e	浮点指数
E	e，但打印大写
f	浮点十进制
F	浮点十进制
g	浮点e或f
G	浮点E或f
%	常量%

□□□

```
>>> x = 1.23456789
>>> x
1.2345678899999999

>>> '%e | %f | %g' % (x, x, x)
'1.234568e+00 | 1.234568 | 1.23457'

>>> '%E' % x
'1.234568E+00'
```

□□□

□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
>>> food = 'spam'
>>> age = 40
>>> vars()
{'food': 'spam', 'age': 40, ...many more... }
```

□□□

□□□□□□□□□□

□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□

```
>>> template = '{0}, {1} and {2}'                    # By position
>>> template.format('spam', 'ham', 'eggs')
'spam, ham and eggs'

>>> template = '{motto}, {pork} and {food}'          # By keyword
>>> template.format(motto='spam', pork='ham', food='eggs')
'spam, ham and eggs'

>>> template = '{motto}, {0} and {food}'             # By both
>>> template.format('ham', motto='spam', food='eggs')
'spam, ham and eggs'
```

□□□

□□□□□□□□□□

□%□□□□□□□□□

□□□□□□□

□□□□%□□□□□□

align□□□□□□□□=□^□□□□□□□□

```
>>> '{0:e}, {1:.3e}, {2:g}'.format(3.14159, 3.14159, 3.14159)
'3.141590e+00, 3.142e+00, 3.14159'

>>> '{0:f}, {1:.2f}, {2:06.2f}'.format(3.14159, 3.14159, 3.14159)


'3.141590, 3.14, 003.14'
```

□□□

□□□Python 2.6□Python 3.0□□□□□□□□□□format□□□□□□□□□□

□%□□□□□□□□

□□□□□□□□

```
# The basics: with % instead of format()

>>> template = '%s, %s, %s'
>>> template % ('spam', 'ham', 'eggs')                    # By position
   'spam, ham, eggs'

>>> template = '%(motto)s, %(pork)s and %(food)s'
>>> template % dict(motto='spam', pork='ham', food='eggs')  # By key
'spam, ham and eggs'

>>> '%s, %s and %s' % (3.14, 42, [1, 2])                  # Arbitrary types
'3.14, 42 and [1, 2]'

# Adding keys, attributes, and offsets

>>> 'My %(spam)s runs %(platform)s' % {'spam': 'laptop', 'platform': sys.platform}
'My laptop runs win32'

>>> 'My %(spam)s runs %(platform)s' % dict(spam='laptop', platform=sys.platform)
'My laptop runs win32'

>>> somelist = list('SPAM')
>>> parts = somelist[0], somelist[-1], somelist[1:3]
>>> 'first=%s, last=%s, middle=%s' % parts
"first=S, last=M, middle=['P', 'A']"
```

□□□

```
'     spam = 123.4567 '

>>> '%(plat)10s = %(item)-10s' % dict(plat=sys.platform, item='laptop')
'    win32 = laptop '

# Floating-point numbers

>>> '%e, %.3e, %g' % (3.14159, 3.14159, 3.14159)
'3.141590e+00, 3.142e+00, 3.14159'

>>> '%f, %.2f, %06.2f' % (3.14159, 3.14159, 3.14159)
'3.141590, 3.14, 003.14'

# Hex and octal, but not binary

>>> '%x, %o' % (255, 255)
'ff, 377'
```

□□□□□□□%□□□□□□□

```
# Building data ahead of time in both

>>> data = dict(platform=sys.platform, spam='laptop')

>>> 'My {spam:<8} runs {platform:>8}'.format(**data)
'My laptop    runs    win32'

>>> 'My %(spam)-8s runs %(platform)8s' % data
'My laptop    runs    win32'
```

□□□□□□□□Python□□□□□□□□□□□□□□□%□□□□□□□□□□□□□□□

```
>>> '{:,d}'.format(999999999999)
'999,999,999,999'

>>> '{:,d} {:,d}'.format(9999999, 8888888)
'9,999,999 8,888,888'

>>> '{:,.2f}'.format(296999.2567)
'296,999.26'
```

□□□□□□□□Python 3.1□□□□□□□□□□□□□□□Python 3.1□□□□ Python 3.0□□□□□□□/□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□ □□□□□□□□□□□□□□Python 3.1□□□□□□□□□□□□□□□□□□24□□□ formats.py□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.1□□□□□ □□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
>>> '{0:b}'.format((2 ** 16) -1)
'1111111111111111'

>>> '%b' % ((2 ** 16) -1)
ValueError: unsupported format character 'b' (0x62) at index 1

>>> bin((2 ** 16) -1)
'0b1111111111111111'

>>> '%s' % bin((2 ** 16) -1)[2:]
'1111111111111111'
```

□□□□□□□□□%□□□□□□□□□

```
C:\misc> C:\Python31\python
>>> 'The {0} side {1} {2}'.format('bright', 'of', 'life')
'The bright side of life'
>>>
>>> 'The {} side {} {}'.format('bright', 'of', 'life')              # Python 3.1+
'The bright side of life'
>>>
>>> 'The %s side %s %s' % ('bright', 'of', 'life')
'The bright side of life'
```

□□□□□Python 3.1□□□□□

```
>>> '%.2f' % 1.2345
'1.23'
>>> '%.2f %s' % (1.2345, 99)
'1.23 99'
```

□□□

□□□

□□□□□□□□□□□

□□□□□□□□□□□□□□□Python□□□□□——□□□□□□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

□□□□

□□□

□□□□

1.□□□find□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□

3.□□□□□□□□□□ASCII□□□□□□□□□□□□□□□□□□□□□

4.□Python□□□□□□□□□□

5.□□□□□S□□□"s,pa,m"□□□□□□□□□□□□□□□□□□□□

6.□□□"a\nb\x1f\000d"□□□□□□□□

7.□□□□□□□string□□□□□□□□□□□□□□□□

□□□□

1.□□□

7.□□□□□□□□string□□□□□□□□□□□□□□□□□□□□string□□□□□□□□
Python 3.0□□□□□□□□□□□□string□□□□□□□□□□□□□□□□□□□

□8□ □□□□□

□□□□

□□□

□8-1□□□

表8-1：常用列表常量和操作（续）

操作	解释
L1 + L2	合并
L * 3	重复
for x in L: print(x)	迭代，成员关系
3 in L	
L.append(4)	方法：增长，排序，搜索，插入，反转等
L.extend([5,6,7])	
L.insert(I, X)	
L.index(1)	
L.Count(X)	
L.sort()	
L.reverse()	
del L[k]	方法，语句：缩短
del L[i:j]	
L.pop()	
L.remove(2)	
L[i:j] = []	
L[i] = 1	索引赋值，分片赋值
L[i:j] = [4,5,6]	
L = [x**2 for x in range(5)]	列表解析（见13章和17章）
list(map(ord, 'spam'))	

□□□

□□□

□□□□□□□□

□□□□□□□

```
>>> str([1, 2]) + "34"                  # Same as "[1, 2]" + "34"
'[1, 2]34'
>>> [1, 2] + list("34")                 # Same as [1, 2] + ["3", "4"]
[1, 2, '3', '4']
```

□□□□□□□

□□□

```
>>> 3 in [1, 2, 3]                          # Membership
True
>>> for x in [1, 2, 3]:
... print(x, end=' ')                       # Iteration
...
1 2 3
```

□□□□□13□□□□□□□□□□□range□□□□□□□□□□□□□□□□□□□

```
>>> res = []
>>> for c in 'SPAM':                        # List comprehension equivalent
...     res.append(c * 4)
...
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

□□□4□□□□□□□□□map□□□□□□□□□

□□□□□□□□

□□□□□□□

```
>>> matrix[1]
[4, 5, 6]
>>> matrix[1][1]
5
>>> matrix[2][0]
7
>>> matrix = [[1, 2, 3],
...           [4, 5, 6],
...           [7, 8, 9]]
>>> matrix[1][1]
5
```

□□□

□□□□□□

□□□□□

1.□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[1]□

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□append□□□□□□□□□□□

□□□

□□□□□□□□□□□□——□□□□□□□□□□□□

```
>>> L = []
>>> L.append(1)                    # Push onto stack
>>> L.append(2)
>>> L
[1, 2]
>>> L.pop()                        # Pop off stack
2
>>> L
[1]
```

pop□□□

□□□□□□□□

□□□□□□□□□□□□□□□del□□□□□□□□□□□□□□□

```
>>> L
['SPAM!', 'eat', 'more', 'please']
>>> del L[0]                              # Delete one item
>>> L
['eat', 'more', 'please']
>>> del L[1:]                             # Delete an entire section
>>> L                                     # Same as L[1:] = []
['eat']
```

□□□□□□□□□□□□□□□

□□□

□□

□□□□□□□□□□□dictionary□□□□Python□□□□□□□□□□□

□□□

表8-2：常见字典常量和操作

操作	解释
`D = {}`	空字典
`D= {'spam': 2, 'eggs': 3}`	两项目字典
`D= {'food': {'ham': 1, 'egg': 2}}`	嵌套
`D= dict.fromkeys(['a', 'b'])`	其他构造技术
`D= dict(zip(keyslist, valslist))`	关键字、对应的对、键列表
`D= dict(name='Bob', age=42)`	
`D ['eggs']`	以键进行索引运算
`D ['food']['ham']`	
`'eggs' in D`	成员关系：键存在测试
`D.keys()`	方法：键
`D.values()`	值
`D.items()`	键+值
`D.copy()`	副本
`D.get(key, default)`	默认

表8-2：常见字典常量和操作（续）

操作	解释
D.update(D2)	合并
D.pop(key)	删除等
len(D)	长度（储存的元素的数目）
D[key] = 42	新增/修改键，删除键
del D [key]	根据键删除条目
list(D.keys())	字典视图（Python 3.0）
D1.keys() & D2.keys()	
Dictionary views (Python 3.0)	
D = {x: x*2 for x in range(10)}	字典解析（Python 3.0）

[1]□□□

□□□□□□□□

□□8-2□□□□□□□

□□len□□□

□□□□□□

□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□value s□items□□□□□□□□□□□□□□□□(key,value)□□□□□□keys□□□□Python 3.0□□□□□□□□□□□list□□□□□□□□□□□□□□□□□

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> list(D.values())
[3, 1, 2]
>>> list(D.items())
[('eggs', 3), ('ham', 1), ('spam', 2)]
```

□□□□□□□□□□□□□□□□□□□

□□□update□□□

□□□□□□□□copy□□□□□□□□

□□□

□□□

□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□for□□□□□keys□□□□□□□□D□□□□□for key in D:□□□□□□key in D.keys():□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

```
>>> L = []
>>> L[99] = 'spam'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list assignment index out of range
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[0]*100□□□□□□□□□□□□□□□□□□□

```
>>> Matrix = {}
>>> Matrix[(2, 3, 4)] = 88
>>> Matrix[(7, 8, 9)] = 99
>>>
>>> X = 2; Y = 3; Z = 4                    # ; separates statements
>>> Matrix[(X, Y, Z)]
88
>>> Matrix
{(2, 3, 4): 88, (7, 8, 9): 99}
```

□□□

```
>>> if (2,3,6) in Matrix:                # Check for key before fetch
...     print(Matrix[(2,3,6)])           # See Chapter 12 for if/else
... else:
...    print(0)
...
0
>>> try:
...     print(Matrix[(2,3,6)])           # Try to index
... except KeyError:                     # Catch and recover
...     print(0)                         # See Chapter 33 for try/except
...
0
>>> Matrix.get((2,3,4), 0)              # Exists; fetch and return
88
>>> Matrix.get((2,3,6), 0)              # Doesn't exist; use default arg
0
```

□□□□□□□□□□□□get□□□□□□□□□□□□

```
>>> rec = {}
>>> rec['name'] = 'mel'
>>> rec['age'] = 45
>>> rec['job'] = 'trainer/writer'
>>>
>>> print(rec['name'])
mel
```

□□□□□□□□□□Python□□□□□□□□□□

□□□

□□□

□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.3□□□□□□□□□□□□□□□dict□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
{'name': 'mel', 'age': 45}              # Traditional literal expression

D = {}                                  # Assign by keys dynamically
D['name'] = 'mel'
D['age'] = 45

dict(name='mel', age=45)                # dict keyword argument form

dict([('name', 'mel'), ('age', 45)])    # dict key/value tuples form
```

□□□□□□□□□□□□□□□□□□□

□□□

Python 3.0□□□□□□

□□□□□□

□□□□Python 2.6□Python 3.0□□□□

```
>>> D = {x: x ** 2 for x in [1, 2, 3, 4]}                # Or: range(1, 5)
>>> D
{1: 1, 2: 4, 3: 9, 4: 16}

>>> D = {c: c * 4 for c in 'SPAM'}                       # Loop over any iterable
>>> D
{'A': 'AAAA', 'P': 'PPPP', 'S': 'SSSS', 'M': 'MMMM'}

>>> D = {c.lower(): c + '!' for c in ['SPAM', 'EGGS', 'HAM']}
>>> D
{'eggs': 'EGGS!', 'ham': 'HAM!', 'spam': 'SPAM!'}
```

□□□

□Python 3.0□□□□□keys□values□items□□□□□□□□□Python 2.6□□□□□□□□□

```
>>> for k in D.keys(): print(k)          # Iterators used automatically in loops
...
a
c
b
```

□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□——□□□Python 2.6□□□□□□□□□□□□□□□□□□□keys□

```
>>> for key in D: print(key)             # Still no need to call keys() to iterate
...
a
c
b
```

□□□□Python 2.X□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> D = {'a':1, 'b':2, 'c':3}
>>> D
{'a': 1, 'c': 3, 'b': 2}

>>> K = D.keys()
>>> V = D.values()
>>> list(K)                              # Views maintain same order as dictionary
['a', 'c', 'b']
>>> list(V)
[1, 3, 2]

>>> del D['b']                           # Change the dictionary in-place
>>> D
{'a': 1, 'c': 3}

>>> list(K)                              # Reflected in any current view objects
['a', 'c']
>>> list(V)                              # Not true in 2.X!
[1, 3]
```

□□□□□□□

□Python 2.X□□□□□□□□□keys□□□□□□Python 3.0□□□□□□□□□

```
>>> D = {'a': 1}
>>> list(D.items())                    # Items set-like if hashable
[('a', 1)]
>>> D.items() | D.keys()               # Union view and view
{('a', 1), 'a'}
>>> D.items() | D                      # dict treated same as its keys
{('a', 1), 'a'}

>>> D.items() | {('c', 3), ('d', 4)}          # Set of key/value pairs
{('a', 1), ('d', 4), ('c', 3)}
>>> dict(D.items() | {('c', 3), ('d', 4)})    # dict accepts iterable sets too
{'a': 1, 'c': 3, 'd': 4}
```

□□□□□□□□□□□□□□□□□□5□□□□□□□□□□Python 3.0□□□□□□3□□□□□□□□□□

□□□□

□□□□□keys□□□□□□□□□□Python 2.X□□□□□□□□□□□□□□□□□□

```
>>> D = {'a':1, 'b':2, 'c':3}
>>> D
{'a': 1, 'c': 3, 'b': 2}

>>> Ks = D.keys()                                # Sorting a view object doesn't work!
>>> Ks.sort()
AttributeError: 'dict_keys' object has no attribute 'sort'

>>> Ks = list(Ks)                                # Force it to be a list and then sort
>>> Ks.sort()
>>> for k in Ks: print(k, D[k])
...
a 1
b 2
c 3

>>> D
{'a': 1, 'c': 3, 'b': 2}
>>> Ks = D.keys()                                # Or you can use sorted() on the keys
>>> for k in sorted(Ks): print(k, D[k])          # sorted() accepts any iterable
...                                             # sorted() returns its result
a 1
b 2
c 3

>>> D
{'a': 1, 'c': 3, 'b': 2}                        # Better yet, sort the dict directly


>>> for k in sorted(D): print(k, D[k])          # dict iterators return keys
...
a 1
b 2
c 3
```

□□□□□□□□□□

□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□

Python 3.0□□□

□□□□

□□□□□□□□□□□□□□□□——□□□□□□Python□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□'a'□'b'□□□□□□□□□□□□□□□□0□

3.□□□□□□□□□□□□□□□□□□□□□□□□□

4.□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□[0,0,0,0,0]□□□□□□□□□□[0]*5□□□□□□□□□□

□9□　□□□□□□□□

□□□□□□□□

□□□□□□□

□□□□□□□

表9-1：常见元组常量和运算

运算	解释
()	空元组
T= (0,)	单个元素的元组（非表达式）
T = (0, 'Ni', 1.2, 3)	四个元素的元组
T = 0, 'Ni', 1.2, 3	另一个四元素的元组（与前列相同）
T = ('abc', ('def', 'ghi'))	嵌套元组
T = tuple('spam')	一个可迭代对象的项的元组
T[i]	索引、索引的索引、分片、长度
T[i][j]	
T[i:j]	
len(T)	
T1 + T2	合并、重复
T * 3	
for x in T: print(x)	迭代、成员关系
'spam' in T	
[x ** 2 for x in T]	

表9-1：常见元组常量和运算（续）

运算	解释
T.index('Ni')	Python 2.6和Python 3.0中的方法：搜索、计数
T.count('Ni')	

□□□□□□□□

□□□□□□□

```
>>> (1, 2) + (3, 4)                # Concatenation
(1, 2, 3, 4)

>>> (1, 2) * 4                     # Repetition
(1, 2, 1, 2, 1, 2, 1, 2)

>>> T = (1, 2, 3, 4)               # Indexing, slicing
>>> T[0], T[1:3]
(1, (2, 3))
```

□□□□□□□□□□□□□□□

□9-1□□□□□□□□□□□□□□

print□□□

□□□□□List comprehension□□□□□

□□□

□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□

□□□

表9-2：常见文件运算

操作	解释
output = open(r'C:\spam', 'w')	创建输出文件（'w'是指写入）
input = open('data', 'r')	创建输入文件（'r'是指读写）
input = open('data')	与上一行相同（'r'是默认值）
aString = input.read()	把整个文件读进单一字符串
aString = input.read(N)	读取之后的N个字节（一或多个）到一个字符串
aString = input.readline()	读取下一行（包括行末标识符）到一个字符串
aList = input.readlines()	读取整个文件到字符串列表
output.write(aString)	写入字节字符串到文件
output.writelines(aList)	把列表内所有字符串写入文件
output.close()	手动关闭（当文件收集完成时会替你关闭文件）
output.flush()	把输出缓冲区刷到硬盘中，但不关闭文件
anyFile.seek(N)	修改文件位置到偏移量N处以便进行下一个操作
for line in open('data'): *use line*	文件迭代器一行一行地读取
open('f.txt', encoding='latin-1')	Python 3.0 Unicode文本文件（str字符串）
open('f.bin', 'rb')	Python 3.0二进制byte文件（bytes字符串）

□□□□

□□□□□□□□□□□□□□□□□□open□□□□□□□□□□□□□□□□□□

·□□“+”□□□

□□□□

□□□

□□□□close□□□□

□□□□□□□□

□□□□□□□

```
>>> open('myfile.txt').read()                    # Read all at once into string
'hello text file\ngoodbye text file\n'

>>> print(open('myfile.txt').read())             # User-friendly display
hello text file
goodbye text file
```

□□□

□□□Python 2.6□□□□□□8□□□□□□

□□□

□□□

```
>>> int(parts[1])                          # Convert from string to int
44
>>> numbers = [int(P) for P in parts]      # Convert all in list at once
>>> numbers
[43, 44, 45]
```

int□□□□□□□□□□□□□□□□□□□□□□□4□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□Python□□□ pickle□□□□□□□□□□□

pickle□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□pickle□□□

□□□□□□□□□□□□□□□□□□□□□□□□'wb'□□□□□□□□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□33□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□Python 2.6□□□□□□□□□□□□□□□□□□□

□□□□□□

□9-2□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□Python□□□□27□□□□□□

Shell□□□

□os.popen□subprocess.Popen□□□□□□□□□□shell□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□PySerial□□□□□□□□□□□□□□□pexpect□□□□□□□□□□□□□□□□□□Python□□□□Web□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□Python 2.5□□□□□□□□□□□□□□open□□□□□□file□□□□□□□□□□□□□□□□□□open□file□□□□□□□□□□□□□□open□□□□□□Python 3.0□□□□file□□□□□□□□□□open□□□

Python 2.6□□□□□□□□□□file□□Python 3.0□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□

表9-3：对象分类

对象类型	分类	是否可变
数字	数值	否
字符串	序列	否
列表	序列	是
字典	对应	是
元组	序列	否
文件	扩展	N/A
Sets	集合	是
frozenset	集合	否
bytearray (3.0)	序列	是

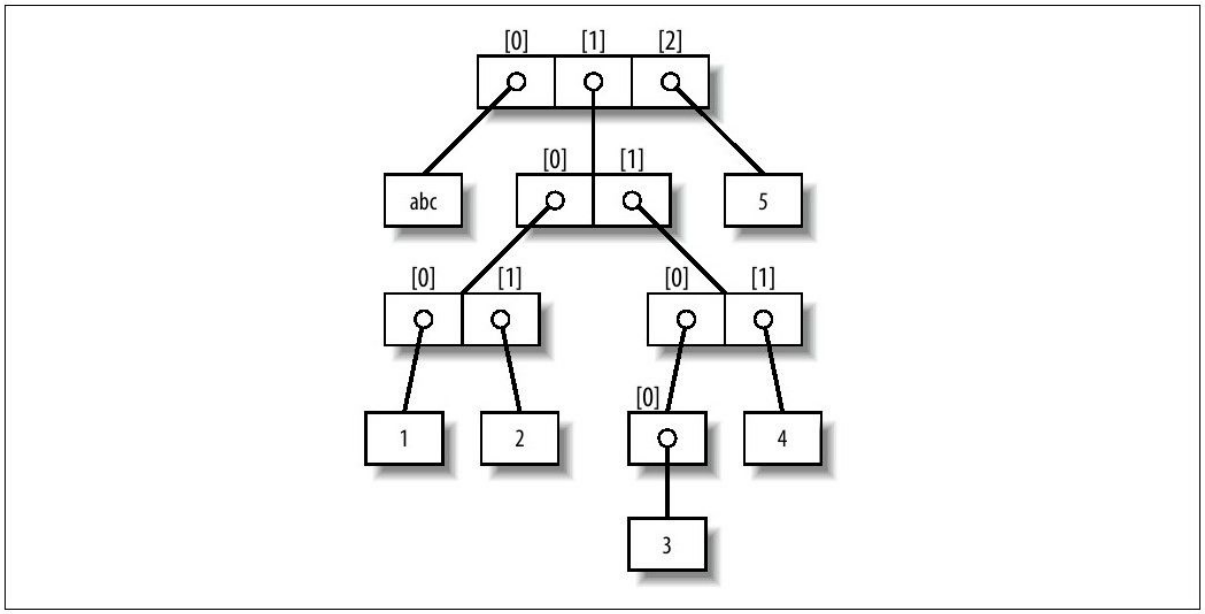
□□□□□□□□□□□□

□□□□□□□□□□□□

□□□

□□□□□

□□□□



□ 9-1 □□□□□□□□□□□['abc',[(1,2),([3],4)],5]□□□□□□□□□□

□□VS□□

□□□□6□□□□

□ 9-2 □□□□□□□□□□□□X□□□□□□□□□L□D□□□□□□□□□□□□X□□□□□□L□D□□□□□□□□□

□□□

```
>>> X[1] = 'surprise'                    # Changes all three references!
>>> L
['a', [1, 'surprise', 3], 'b']
>>> D
{'x': [1, 'surprise', 3], 'y': 2}
```

□□□□□□□□□□□□□□□□□□□

□□□

copy□□——□□import copy□□□□□□X=copy.deepcopy(Y)□□□□□□□Y□□□□□□□

□□□□□□□□□

□□□Python□□□□□□□□□□□□□——□□□□□□□□□□□□□Python□□□□□□□□□□□□□

```
>>> S1 = 'spam'
>>> S2 = 'spam'
>>> S1 == S2, S1 is S2
(True, True)
```

□□□“==”□□□□□□□is□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>> L1 = [1, ('a', 3)]
>>> L2 = [1, ('a', 2)]
>>> L1 < L2, L1 == L2, L1 > L2                    # Less, equal, greater: tuple of results
(False, False, True)
```

□□□□□3□□2□□□□L1□□L2□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python
>>> D1 = {'a':1, 'b':2}
>>> D2 = {'a':1, 'b':3}
>>> D1 == D2
False
>>> D1 < D2
TypeError: unorderable types: dict() < dict()

>>> list(D1.items())
[('a', 1), ('b', 2)]
>>> sorted(D1.items())
[('a', 1), ('b', 2)]

>>> sorted(D1.items()) < sorted(D2.items())
True
>>> sorted(D1.items()) > sorted(D2.items())
False
```

□□□

Python□□□□□□□

□□□□□□□

表 9-4：对象真值的例子

对象	值
"spam"	True
""	False
[]	False
{}	False
1	True
0.0	False
None	False

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□if X:□□□□□□□□□□□□□□X□□□□□□□□□□□□□□□if X !='':□□□□□□□□□□□□□□□□□□□

□□□

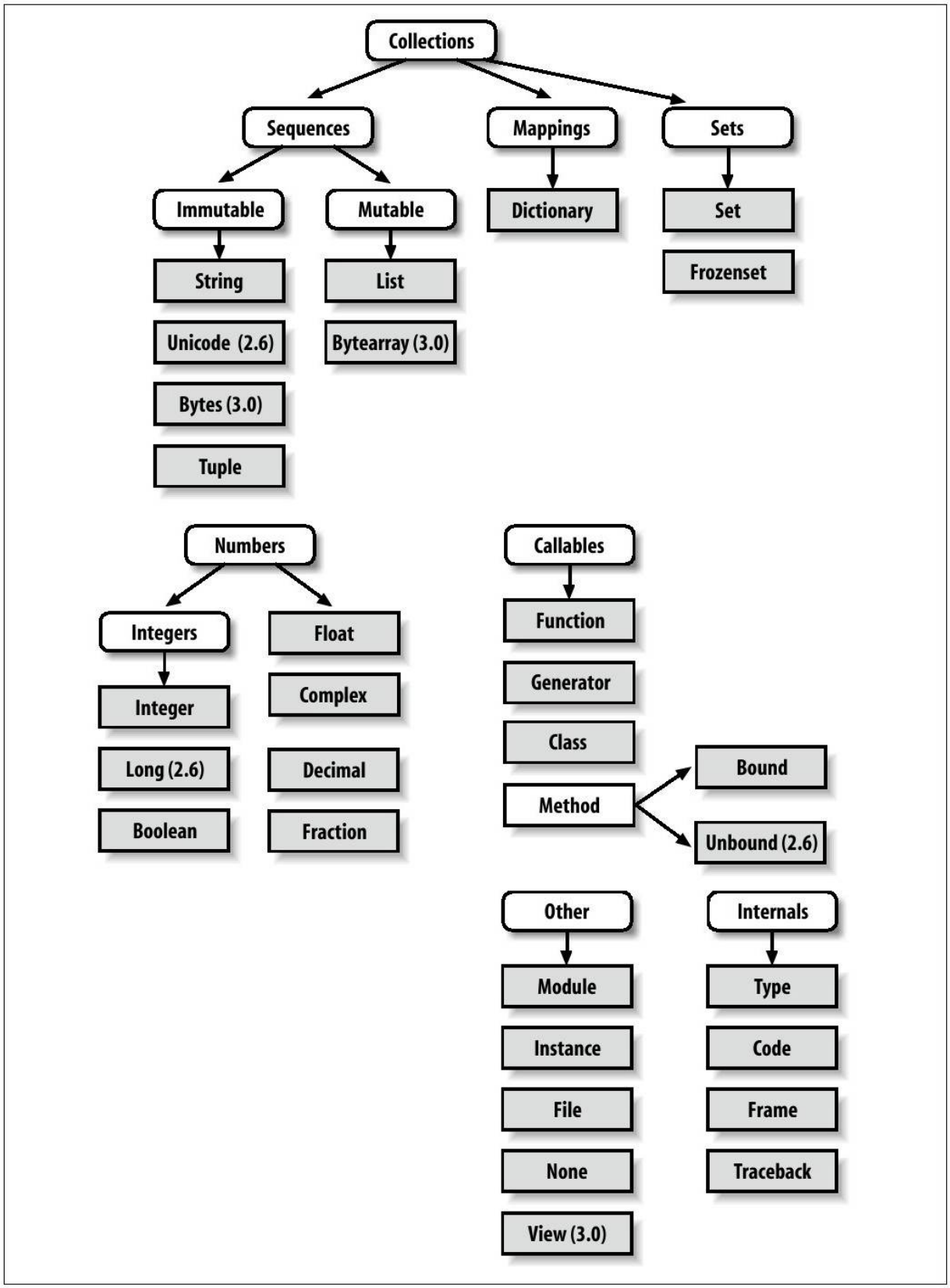
·□□□□□□□□□□□□□□□True□False□□□□□□□□1□0□□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□True□False□□□□□□□□1□0□□□□□□□□□□□□□□□□□□□□□□□□

□if□□□□□□□□□□□□□□□□□□□

Python□□□□□

□9-3□□□Python□□□□□


Collections
Sequences
Mappings
Sets
Immutable
Mutable
Dictionary
Set
String
List
Frozenset
Unicode (2.6)
Bytearray (3.0)
Bytes (3.0)
Tuple
Numbers
Callables
Integers
Float
Function
Integer
Complex
Generator
Long (2.6)
Decimal
Class
Boolean
Fraction
Method
Bound
Unbound (2.6)
Other
Internals
Module
Type
Instance
Code
File
Frame
None
Traceback
View (3.0)

□ 9-3 □□□□□□Python□□□□□□□□□□Python□□□□□□□□□□

Type□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□

□□□□□□□□□□type(X)□□□□□□X□□□□□□□□□□□□□□□Python□□if□□□□□□□□□□□□□□□□4□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□Python 2.2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□dict□list□str□tuple□int□float□complex□byte□type□set□file□□Python 2.6□file□□□□□□□□open□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□

□□□□Python□□□□

Python□□□□□□

□□□□□□

□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□

□□□□

□□□

□□□□

1.□□□□□□□□□□□□

2.□□□□□□□□□□□□

□□□□

1.□□□len□□□□□Python□□□□□

6.□□□

□□□□ □□□

□□□

```
  2 ** 16
  2 / 5, 2 / 5.0

  "spam" + "eggs"
  S = "ham"
  "eggs " + S
  S * 5
  S[:0]
  "green %s and %s" % ("eggs", S)
  'green {0} and {1}'.format('eggs', S)

  ('x',)[0]
  ('x', 'y')[1]

  L = [1,2,3] + [4,5,6]
  L, L[:], L[:0], L[-2], L[-2:]
  ([1,2,3] + [4,5,6])[2:4]
  [L[2], L[3]]
  L.reverse(); L
  L.sort(); L
  L.index(4)

  {'a':1, 'b':2}['b']
  D = {'x':1, 'y':2, 'z':3}
  D['w'] = 0
  D['x'] + D['w']
  D[(1,2,3)] = 4

list(D.keys()), list(D.values()), (1,2,3) in D

[[]], ["",[],(),{},None]
```

2.□□□

□□□□□□□□□□□□□□□□X□Y□□□□□□□□□

5.□□□□□□□□□□□□□□□

```
>>> D = {}
>>> D[1] = 'a'
>>> D[2] = 'b'
```

□□□□□□□□□□□□□□□□

c.append□□□□□□□□□□□□□□□□□□□□□□□□□□keys□□□□□□□□append□□□□□□□□□□□□□□□□

d.□□

8.□□□□□□□□□□□□□□S□□□□□□□□□□□□S="spam"□□□□□□□□□□□□S[0][0][0][0][0]□□□□□□□□□□□□□□□□□□□

□□□□　□□□□□

□10□　Python□□□□

□□□□□□□□□□Python□□□□□□□□□□

Python□□□

表10-1：Python语句

语句	角色	例子
赋值	创建引用值	`a, b, c = 'good', 'bad', 'ugly'`
调用	执行函数	`log.write("spam, ham")`
打印调用	打印对象	`print ('The Killer', joke)`
if/elif/else	选择动作	`if "python" in text:` `  print (text)`
for/else	序列迭代	`for x in mylist:` `  print (x)`
while/else	一般循环	`while X > Y:` `  print ('hello')`
pass	空占位符	`while True:` `  pass`
break	循环退出	`while True:` `  if exittest(): break`
continue	循环继续	`while True:` `  if skiptest(): continue`
def	函数和方法	`def f(a, b, c=1, *d):` `  print(a+b+c+d[0])`
return	函数结果	`def f(a, b, c=1, *d):` `  return a+b+c+d[0]`
yield	生成器函数	`def gen(n):` `  for i in n: yield i*2`

表10-1：Python语句（续）

语句	角色	例子
global	命名空间	x = 'old' def function(): global x, y; x = 'new'

nonlocal	Namespaces(3.0+)
	命名空间（Python 3.0及其以上版本）
		def outer():
		    x = 'old'
		    def function():
		        nonlocal x; x = 'new'

import	模块访问	import sys
from	属性访问	from sys import stdin
class	创建对象	class Subclass(Superclass): staticData = [] def method(self): pass
try/except/ finally	捕捉异常	try: action() except: print ('action error')
raise	触发异常	raise EndSearch(location)
assert	调试检查	assert X > Y, 'X too small'
with/as	环境管理器（2.6）	with open('data') as myfile: process(myfile)
del	删除引用	del data[k] del data[i:j] del obj.attr del variable

□10-1□□□□Python 3.0□□□□

·□□□□□□print□Python 3.0□□□□□□

□□if□□□

□□□□□□□□10-1□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□C□□□□□□□□□□□Python□□□□□□□□

Python□□□□□

Python□□□□□□□□□□□:□□□□Python□□□□□□□□□□□□□□□□□□□

Python□□□□□

□□Python□□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

```
if (x < y)
```

□□□C□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
if x < y
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□C□□□□□□□□□□□□□□

```
x = 1;
```

□Python□□□□□□□□□□□□□□□□□□□

```
if (x > y) {
    x = 1;
    y = 2;
}
```

□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
if x > y:
    x = 1
    y = 2
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□C□□□□□□□□□

```
while (x > 0) {
    --------;
    --------;
```

□□□

□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□□

```
if (x)
     if (y)
          statement1;
else
    statement2;
```

□□else□□□□□if□□□□□□□□□□□□□else□□□□□□if□□[if(y)]□□□□□□□□□□□□□□□□if(x)□□□□C□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□Python□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□□□□□□□□Python□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□

```
X = A + B + \                          # An error-prone alternative
      C + D
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□Python□□□□□□□□□

□□□□□□□□□□□□□□□□□True□□□□□□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□□□□□□□input□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□raw_input□□□input□□□□□□□□print□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□if□□□"stop"□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
Enter text:xxx
...error text omitted...
ValueError: invalid literal for int() with base 10: 'xxx'
```

□□□□□□□□□□□int□□□□□□□□□□□□□□□□□□□

```
while True:
    reply = input('Enter text:')
    if reply == 'stop':
        break
    elif not reply.isdigit():
        print('Bad!' * 8)
    else:
        print(int(reply) ** 2)
print('Bye')
```

□□□□□12□□□□□□if□□□□□□

□try□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□try□except□□else□□□□□□□□□□□□□

□□□□□□

□□□□□

```
Enter text:19
low
Enter text:20
400
Enter text:spam
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Enter text:stop
Bye
```

□□□□

□□□□□□□□□□Python□□□□□□□□□□

□□□□

1.□C□□□□□□□□□□Python□□□□□□□□□□□

2.Python□□□□□□□□□□□□□□

3.□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□

6.□□□□□□□□Python□□□□□□□□□□

7.try□□□□□□□□□□

8.Python□□□□□□□□□□□□□□□□□□□

□□□□

1.□C□□□

第11章　□□□□□□□□□

□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

·□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□“=”□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□'lunch'□□□□□□□□□□□□□□□ham='lunch'□□□□□□spam=ham□□□□□□ham□□□□□□□□□□□□□□□

□□□□

□□□

□□□□Python□□□□

```
>>> a, b, c = string[0], string[1], string[2:]          # Index and slice
>>> a, b, c
('S', 'P', 'AM')

>>> a, b, c = list(string[:2]) + [string[2:]]          # Slice and concatenate
>>> a, b, c
 ('S', 'P', 'AM')

>>> a, b = string[:2]                                  # Same, but simpler
>>> c = string[2:]
>>> a, b, c


 ('S', 'P', 'AM')

 >>> (a, b), c = string[:2], string[2:]                # Nested sequences
 >>> a, b, c
 ('S', 'P', 'AM')
```

□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□13□□□□□□□□□for□□□□□□□□□

```
>>> L = [1, 2, 3, 4]
>>> while L:
...     front, L = L[0], L[1:]                    # See next section for 3.0 alternative
...     print(front, L)
...
1 [2, 3, 4]
2 [3, 4]
3 [4]
4 []
```

□□□

Python 3.0□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□ Python 2.6□□□□□□□□□□□□□□□□□□□

```
>>> a, *b = seq
>>> a
1
>>> b
[2, 3, 4]
```

□□□

□□□

```
>>> S = 'spam'

>>> S[0], S[1:]                    # Slices are type-specific, * assignment always returns a list
('s', 'pam')

>>> S[0], S[1:3], S[3]
('s', 'pa', 'm')
```

Python 3.0□□□

```
>>> seq
[1, 2, 3, 4]

>>> a, b, c, *d = seq
>>> print(a, b, c, d)
1 2 3 [4]
```

□□□

```
>>> a, *b, c, *d = seq
SyntaxError: two starred expressions in assignment

>>> a, b = seq
ValueError: too many values to unpack

>>> *a = seq
SyntaxError: starred assignment target must be in a list or tuple

>>> *a, = seq
>>> a
[1, 2, 3, 4]
```

□□□□□□□□□

□□□□□□□□

```
>>> *a, b = seq                          # Rest, last
>>> a, b
([1, 2, 3], 4)

>>> a, b = seq[:-1], seq[-1]             # Rest, last: traditional
>>> a, b
([1, 2, 3], 4)
```

□□□

□□□□□□□□□□□□□□□

```
for (a, b, c) in [(1, 2, 3), (4, 5, 6)]:       # a, b, c = (1, 2, 3), ...
```

□□□□□□□Python 2.6□□□□□□□□□□Python 3.0□□□□□□□□□

```
for all in [(1, 2, 3, 4), (5, 6, 7, 8)]:
    a, b, c = all[0], all[1:3], all[3]
```

□□13□□□□□□□□□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

```
>>> a = b = 0
>>> b = b + 1
>>> a, b
(0, 1)
```

□□□□□□b□□□b□□□□□□

□□□□□□

□Python 2.0□□□11-2□□□□□

□□□

```
>>> L = [1, 2]
>>> L = L + [3]                    # Concatenate: slower
>>> L
[1, 2, 3]

>>> L.append(4)                    # Faster, but in-place
>>> L
[1, 2, 3, 4]
```

□□□

□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□“+=”□□□□□□□□□□□□□□□□□□□□□□□□□□□□“+”□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

表11-3：Python 3.0中的保留字

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

□11-3□□Python 3.0□□Python 2.6□□□□□□□□□□□□□□

·print□□□

·exec□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·nonlocal□□□□□□□□□□□□□□□□□□□□

□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□

·with□as□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·yield□Python 2.3□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·□Python 2.5□□yield□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□Python□□□□□□□□□□

□□□□□Python 2.6□□with□as□□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□PEP 8□□□□□□□□□□□□□□□□□□□

□□□□□

□Python□□□□

表11-4：常见Python表达式语句

运算	解释
spam(eggs, ham)	函数调用
spam.ham(eggs)	方法调用
spam	在交互模式解释器内打印变量
print(a, b, c, sep='')	Python 3.0中的打印操作
yield x ** 2	产生表达式的语句

□11-4□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□

□□□□□Python□□□□□□□□□□□□

□□□□

□Python□□print□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□Python 3.0□Python 2.6□□□

Python 3.0□□□□□

□□□□

□□□□□□□□□Python 3.0□print□□□□□□□□□

```
print([object, ...][, sep=' '][, end='\n'][, file=sys.stdout])
```

□□□□□□□□□

·file□□□

□□□

```
>>> print(x, y, z, sep='')                  # Suppress separator
spam99['eggs']
>>>
>>> print(x, y, z, sep=', ')                # Custom separator
spam, 99, ['eggs']
```

□□□□□□print□□□□□□□□□□□□□□□□□□□□□□□□□□end□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□file□□□——□□□□□□□□□□

Python 2.6 print□□

□□□□□□□□□Python 2.6□□□□□□□□□

□□Python 2.6 print□□□Python 3.0□□□

□□□□□□□□□□□□□□□□□□□□□□Python 2.6□print□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□

□□□□□□

□Python 3.0□Python 2.6□□□□□

```
>>> import sys                              # Printing the hard way
>>> sys.stdout.write('hello world\n')
hello world
```

□□□□□□□□□sys.stdout□write□□□□Python□□□□□□□□□

```
import sys
sys.stdout = open('log.txt', 'a')           # Redirects prints to a file
...
print(x, y, x)                              # Shows up in log.txt
```

□□□□□□□sys.stdout□□□□□□□

```
C:\misc> c:\python30\python
>>> import sys
>>> temp = sys.stdout                      # Save for restoring later
>>> sys.stdout = open('log.txt', 'a')      # Redirect prints to a file
>>> print('spam')                          # Prints go to file, not here
>>> print(1, 2, 3)
>>> sys.stdout.close()                     # Flush output to disk
>>> sys.stdout = temp                      # Restore original stream

>>> print('back here')                     # Prints show up here again
back here
>>> print(open('log.txt').read())          # Result of earlier prints
spam
1 2 3
```

□□□

```
C:\misc> c:\python30\python
>>> log = open('log.txt', 'w')
>>> print(1, 2, 3, file=log)                    # 2.6: print >> log, 1, 2, 3
>>> print(4, 5, 6, file=log)
>>> log.close()
>>> print(7, 8, 9)                              # 2.6: print 7, 8, 9
7 8 9
>>> print(open('log.txt').read())
1 2 3
4 5 6
```

□□print□□□□□□□□□□□□□□□□□□□□□□□□□□sys.stderr□□□□□□□□□□write□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> import sys
>>> sys.stderr.write(('Bad!' * 8) + '\n')
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!

>>> print('Bad!' * 8, file=sys.stderr)          # 2.6: print >> sys.stderr, 'Bad' * 8
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□print□□□□□□Python 3.0□2to3□□□□□□□□□□□□Python 3.0□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□Python 2.X□□□□□□□Python 3.0□□□□

□□□□□□Python 2.6□□□□□Python 3.0□print□□□□□□□□□□□□□□□□□□□□□□□□

```
from __future__ import print_function
```

□□□□□Python 2.6□□□□□Python 3.0□print□□□□□□□□□□□□□□□□□□Python 3.0 print□□□□□□□□□□□□□Python 3.0□□□□□□print□

□□□□□□□□□□□□□□□□11-5□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python
>>> print('spam')                    # 3.0 print function call syntax
spam
>>> print('spam', 'ham', 'eggs')     # These are mutiple argments
spam ham eggs
```

□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python26\python
>>> print('spam')                    # 2.6 print statement, enclosing parens
spam
>>> print('spam', 'ham', 'eggs')     # This is really a tuple object!
('spam', 'ham', 'eggs')
```

□□□□□□□□□□□□□□□□□□□

□□□□□□print□stdout

print□□□sys.stdout□□□□□□□□□□□□□□□□□□□□□sys.stdout□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□write□□□□print□□□□□□□□□□sys.stdout.write□□□□□□sys.stdout□□□□□□□□□□□□□□□□write□□□□□□□□□□□□□□□□□□□

```
myobj = FileFaker()                    # 3.0: Redirect to object for one print
print(someObjects, file=myobj)         # Does not reset sys.stdout

myobj = FileFaker()                    # 2.6: same effect
print >> myobj, someObjects            # Does not reset sys.stdout
```

Python□□□raw_input()□□□□sys.stdin□□□□

□□□□

□□Python□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3.L=L.sort()□□□□□□□

4.□□□□print□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□A=B=C=0□□□□□□□□□□A,B,C=0,0,0□□□□□□□□□□□□□□□A=0,B=0,C=0□□□□□□□□□□□□□□□□10□□A=0;B=0;C=0□□

2.□□□□□□□□□□□□

```
A = B = C = []
```

□□□□□□□□□□□□□□□□□□□

□□□

□12□　if□□□□□□□

□□□□Python□if□□□□□□□

```
if <test1>:                          # if test
    <statements1>                    # Associated block
elif <test2>:                        # Optional elifs
    <statements2>


  else:                                # Optional else
      <statements3>
```

□□□□

□□□□□□□□□□□□if□□□□□□□□□□□□if□□□□□□□□□□□□

□□□□

□□□□□□□□if□□□□□□□□□□□□□□□□□□

```
>>> x = 'killer rabbit'
>>> if x == 'roger':
...     print("how's jessica?")
... elif x == 'bugs':
...     print("what's up doc?")
... else:
...     print('Run away! Run away!')
...
Run away! Run away!
```

□□□□□□□if□□□□else□□□□□□Python□□□□□□□□□□□□□□□□□□

```
>>> choice = 'ham'
>>> print({'spam': 1.25,                    # A dictionary-based 'switch'
...        'ham': 1.99,                     # Use has_key or get for default
...        'eggs': 0.99,
...        'bacon': 1.10}[choice])
1.99
```

□□□

```
>>> branch = {'spam': 1.25,
...           'ham': 1.99,
...           'eggs': 0.99}
>>> print(branch.get('spam', 'Bad choice'))
1.25
>>> print(branch.get('bacon', 'Bad choice'))
Bad choice
```

□□□□if□□□□in□□□□□□□□□□□□□□□

```
>>> choice = 'bacon'
>>> if choice in branch:
...     print(branch[choice])
... else:
...     print('Bad choice')
...
Bad choice
```

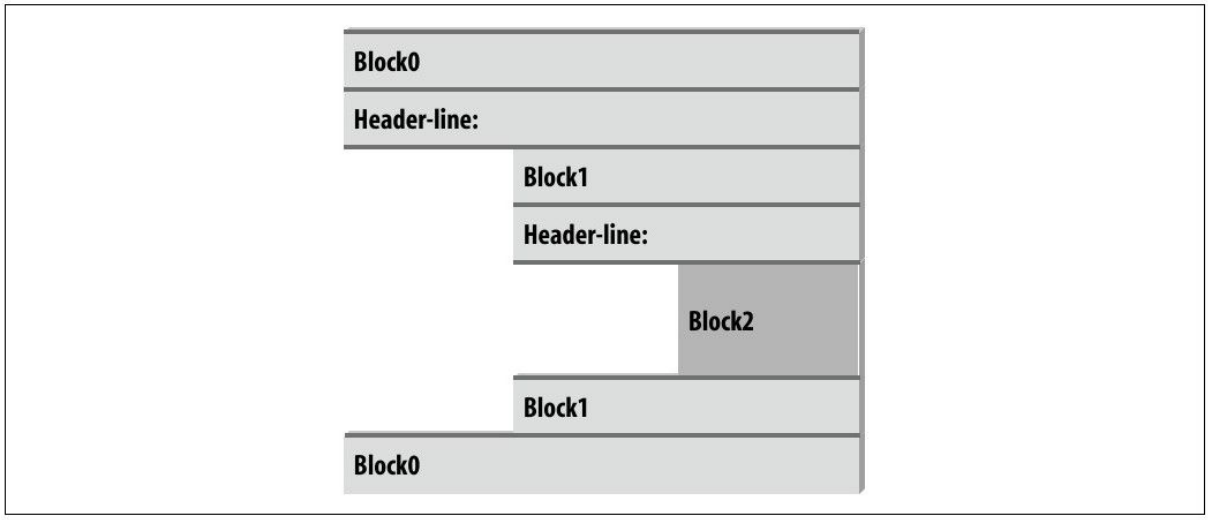
□□□□□□□□□□□□□□□□□□□□□□if□□□□□□□□□□□□□□□□□□□

Python□□□□

□10□□□□Python□□□□□□□□□□□□□□□□□□□if□□□□□□□□□□□□□□□□□□□

·□□□□□□□docstring□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
x = 1
if x:
    y = 2
    if y:
        print('block2')
    print('block1')
print('block0')
```



□ 12-1 □□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

Python□□□□□

□□□□□□#□□□□□□□□□□□□□□□□□□

□□□□□□

□□

```
L = ["Good",
     "Bad",
     "Ugly"]                          # Open pairs may span lines
```

□□□□□□□□□□□□□□□□□□□

□□□

```
S = """
aaaa
bbbb
cccc"""

S = ('aaaa'
     'bbbb'                                  # Comments here are ignored
     'cccc')
```

□□□Python□□□

□□□□

□□□

not X

□□X□□□□□□□□□□□□□□□True□False□□

□□□X□Y□□□□□□□□□□□□□□□

```
>>> 2 or 3, 3 or 2          # Return left operand if true
(2, 3)                      # Else, return right operand (true or false)
>>> [] or 3
3
>>> [] or {}
{}
```

□□□□□□□□□□□□2□3□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□

and□or□□□

if/else□□□□□

Python□□□□□

```
>>> A = 't' if 'spam' else 'f'                    # Nonempty is true
>>> A
't'
>>> A = 't' if '' else 'f'
>>> A
'f'
```

Python 2.5□□□□□2.5□□□□□

```
>>> ['f', 't'][bool('')]
'f'
>>> ['f', 't'][bool('spam')]
't'
```

□□□□□□□□□□□□□□Python□□□□□□□□□□□X□□□□□□□□□□□Z□Y□□□□□□□□□□Python 2.5□□□□□□□□□□□□□□□□□□if/else□□□□□□□□□□□□□□□□□□

```
X = A or default
```

□□□

□□□

□□□□

□□□□□□□□□□Python□if□□□□□□□□□□

□□□□

1.□Python□□□□□□□□□□□

2.□Python□□□□if/else□□□□□□□□

3.□□□□□□□□□□□□□

4.True□False□□□□□□□□□□□□

□□□□

1.if□□□□□elif□□□□□

第13章　while和for循环

循环结构是程序设计中必不可少的一种结构，Python语言中的循环结构有两种，即while循环和for循环，本章将介绍这两种循环结构的语法和使用方法。

循环结构的特点是，在给定条件成立时，反复执行某个程序段，被反复执行的程序段称为循环体。循环结构中还有break和continue两个保留字，用于辅助循环结构的执行。此外，range、zip、map等函数也常用于循环结构中。

本章首先介绍while和for循环的基本语法，然后介绍Python语言中的一些特殊用法，包括Python语言中的for循环的一些特殊用法，for循环与列表推导式，for循环与filter、reduce函数，以及循环结构的嵌套。

while循环

while循环是Python语言中最基本的循环结构，它的特点是“当”某个条件成立时，反复执行某个程序段，直到条件不成立为止。while循环的语法结构如下：

□□□□

while□□□

□□

□□□□□□□□□□□□□□□□□while□□□□□□□□□□□□□while□□□□□□print□□□□□□□□□□□□□□□□□□□□True□□□□1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> while True:
...     print('Type Ctrl-C to stop me!')
```

□□□□□□□□□□□□□□□□□□□

```
>>> a=0; b=10
>>> while a < b:                        # One way to code counter loops
...     print(a, end=' ')
...     a += 1                          # Or, a = a + 1
...
0 1 2 3 4 5 6 7 8 9
```

□□□Python□□□□□□□□□□□"do until"□□□□□□□□□□□□□□□□□□□□□□□□□□□□□break□□□□□□□□□

```
while True:
    ...loop body...

        if exitTest(): break
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□break□□□

break、continue、pass、以及else

□□□□□□□□□□Python□□□□□□□□□□

□□□□□□

□□break□continue□□□□while□□□□□□□□□□□□

```
while <test1>:
   <statements1>
   if <test2>: break          # Exit loop now, skip else
   if <test3>: continue       # Go to top of loop now, to test1
else:
   <statements2>              # Run if we didn't hit a 'break'
```

break□continue□□□□□while□□for□□□□□□□□□□□□

pass

pass□□□

□□□

continue

continue□□□

break

break□□□□□□□□□□□□□□□break□□□□□□□□□□□□□□□

□□else

□□□else□□□□□□□break□□□□□□□

□□□□else□□□□□□□

□□□□else□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□else□□□□□□□□□□□□□□□□□□□

```
while x:                                    # Exit when x empty
    if match(x[0]):
        print('Ni')
        break                               # Exit, go around else
    x = x[1:]
else:
    print('Not found')                       # Only here if exhausted x
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□else□□while□□□□□□□□□□□□□□□if□□□□□□while□□□□break□□□□□□□□□else□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□else□□□□□□□□□□□□x□□□□□□□□□□if not x:□□□□□□□□□□□□□□□□else□□□□□□□□□□□□□□□□□□□

□□□□□□□□□Python while□□□□□□□□□

for□□

for□□□Python□□□for□□□
□□□□□□□□□□□□□□□□□□□

□□□□

Python for□□□

```
for <target> in <object>:                # Assign object items to target
    <statements>
    if <test>: break                     # Exit loop now, skip else
    if <test>: continue                  # Go to top of loop now
else:
    <statements>                         # If we didn't hit a 'break'
```

□□

□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□x□□□□□□□□□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□□x□□□□□□□□□□□□□□□

```
>>> for x in ["spam", "eggs", "ham"]:
...     print(x, end=' ')
...
spam eggs ham
```

□□□□□□□□□□□□□□□□□□□

```
10
>>> prod = 1
>>> for item in [1, 2, 3, 4]: prod *= item
...
>>> prod
24
```

□□□□□□

□□□□□□□for□□□□□□□□□□□□□□□for□□□□□□□□□□□□

```
>>> S = "lumberjack"
>>> T = ("and", "I'm", "okay")

>>> for x in S: print(x, end=' ')                    # Iterate over a string
...
l u m b e r j a c k

>>> for x in T: print(x, end=' ')                    # Iterate over a tuple
...
and I'm okay
```

□□□□□□□□□□□□□for□□

□for□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```
>>> T = [(1, 2), (3, 4), (5, 6)]
>>> for (a, b) in T:                          # Tuple assignment at work
...     print(a, b)
...
1 2
3 4
5 6
```

□□□□□□□□□□□□□□□□□□(a,b)=(1,2)□□□□□□□□□□□(a,b)=(3,4)□□□□□□□□□□□□□□□□□□

□□for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□

```
1 2 3
X Y 6
```

Python 3.0□for□□□□□□□□□□

□□□□□□for□□Python 3.0□□□□□□□□□□□□□□□□□□□

```
>>> a, *b, c = (1, 2, 3, 4)                          # Extended seq assignment
>>> a, b, c
(1, [2, 3], 4)

>>> for (a, *b, c) in [(1, 2, 3, 4), (5, 6, 7, 8)]:
...     print(a, b, c)
...
1 [2, 3] 4
5 [6, 7] 8
```

□□□

```
>>> items = ["aaa", 111, (4, 5), 2.01]          # A set of objects
>>> tests = [(4, 5), 3.14]                      # Keys to search for
>>>
>>> for key in tests:                           # For all keys
...     for item in items:                      # For all items
...         if item == key:                     # Check for match
...             print(key, "was found")
...             break
...     else:
...         print(key, "not found!")
...
(4, 5) was found
3.14 not found!
```

□□□□□□□□if□□□□□□□□□□□□□break□□□□else□□□□□□□□□□□□□

□□□□□□for□□□□□□

```
file = open('test.txt')
while True:
    char = file.read(1)                    # Read by character
    if not char: break
    print(char)

for char in open('test.txt').read():
    print(char)
```

□□□for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□while□□□□□□□□□□□□□□□□□□□

□□readlines□□□

□□□□□□□

for□□□□□□□□□□□□□□□□□□□□□□□□for□while□□□□□□□□□□□□□□□□□□□

```
>>> list(range(5)), list(range(2, 5)), list(range(0, 10, 2))
([0, 1, 2, 3, 4], [2, 3, 4], [0, 2, 4, 6, 8])
```

□□□□□□range□□□□□□

```
>>> X = 'spam'
>>> for item in X: print(item, end=' ')                    # Simple iteration
...
s p a m
```

□□□□□□□□□□for□□□□□□□□□□

□□□□□□□□□□X□□□□□□□□□□□□X□□□□□□□□□□□□□□□□X□□□□□□□□□□□□□□□□

□□□□□□range□□□

□□□□□

□□□□□□□□□2□□□□

```
>>> S = 'abcdefghijk'
>>> for c in S[::2]: print(c, end=' ')
...
a c e g i k
```

□□□□□□□□□

□□□□□range

□□□□range□for□□□□

```
>>> L = [1, 2, 3, 4, 5]

>>> for i in range(len(L)):                    # Add one to each item in L
...     L[i] += 1                              # Or L[i] = L[i] + 1
...
>>> L
[2, 3, 4, 5, 6]
```

□□□

□□□□□zip□map

□□□□□□□□□□□□□□□range□□□□□for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□zip□□□□□□□□□for□□□□□□□□□□□□□□□□□□□□□zip□□□□□□□□□□□□□□□□□□□

```
>>> for (x, y) in zip(L1, L2):
...     print(x, y, '--', x+y)
...
1 5 -- 6
2 6 -- 8
3 7 -- 10


 4 8 -- 12
```

□□□□□□□□□□□□zip□□□□□□□□□□□□

```
>>> T1, T2, T3 = (1,2,3), (4,5,6), (7,8,9)
>>> T3
(7, 8, 9)
>>> list(zip(T1, T2, T3))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

□□□□□□□□□zip□□□□□□□□□

□□□□19□□□20□□□□□□□□map□□□□□□□□

```
>>> D1 = {'spam':1, 'eggs':3, 'toast':5}
>>> D1
{'toast': 5, 'eggs': 3, 'spam': 1}

>>> D1 = {}
>>> D1['spam'] = 1
>>> D1['eggs'] = 3
>>> D1['toast'] = 5
```

□□□

□□□□Python 2.2□□□□□□□□□□□□□□□□for□□□□□□zip□□□/□□□□□□□□dict□□□□□

```
>>> keys = ['spam', 'eggs', 'toast']
>>> vals = [1, 3, 5]

>>> D3 = dict(zip(keys, vals))
>>> D3
{'toast': 5, 'eggs': 3, 'spam': 1}
```

□□□□□dict□□□Python□□□□□□□□□31□□□□□□□□□□□□□□□□

□□□□□□□□enumerate

□□□□□□□□□□range□□□□□□□□□□

```
>>> S = 'spam'
>>> for (offset, item) in enumerate(S):
...     print(item, 'appears at offset', offset)
...
s appears at offset 0
p appears at offset 1
a appears at offset 2
m appears at offset 3
```

enumerate□□□

```
>>> [c * i for (i, c) in enumerate(S)]
['', 'p', 'aa', 'mmm']
```

□□enumerate□zip□□□

□□□□

□□□□□□□□□□Python□□□□□□□□□□Python□□□□□□□□□□□□□□□□□while□for□□□□□□□□□□□else□□□□□□□□□break□continue□□□□□□□□□□□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□range□zip□map□enumerate□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□

□□□□

1.While□for□□□□□□□□□□□□□□

2.break□continue□□□□□□□

3.□□□□□else□□□□□□□

4.□Python□□□□□□□□□□□□□□□□□□

5.□□□range□□for□□□□

□□□□

1.while□□□□□□□□□□□□□□for□□□□□□□□□□□□□□

第14章　□□□□□□□□□□□□

□□□□□□□□□□□□Python□□□□□□□□while□for□□□□□□□□□□□□

```
>>>for x in [1, 2, 3, 4]: print(x ** 2, end=' ')
...
1 4 9 16

>>>for x in (1, 2, 3, 4): print(x ** 3, end=' ')
...
1 8 27 64

>>>for x in 'spam': print(x * 2, end=' ')
...
ss pp aa mm
```

□□□□for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>>f = open('script1.py')                # Read a 4-line script file in this directory
>>>f.readline()                          # readline loads one line on each call
'import sys\n'
>>>f.readline()
'print(sys.path)\n'
>>>f.readline()
'x = 2\n'
>>>f.readline()
'print(2 ** 33)\n'
>>>f.readline()                          # Returns empty string at end-of-file
''
```

□□□□□□□□□□□□□□□□__next__□□□□□□□□□□□□□□□□

□□□9□□□

□□□

□□□□□iter□next

□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□next□□□□□□□□□□□□□__next__□□□□□□□□□□□□□X□□□next(X)□□□X.__next__()□□

```
>>>f = open('script1.py')
>>>f.__next__()                          # Call iteration method directly
'import sys\n'
>>>f.__next__()
'print(sys.path)\n'

>>>f = open('script1.py')
>>>next(f)                               # next built-in calls __next__
'import sys\n'
>>>next(f)
'print(sys.path)\n'
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□iter□□□□□□□□□□□□□□□□□□□

```
>>>L = [1, 2, 3]
>>>I = iter(L)                    # Obtain an iterator object
>>>I.next()                       # Call next to advance to next item
1
>>>I.next()
2
>>>I.next()
3
>>>I.next()
Traceback (most recent call last):
...more omitted...
StopIteration
```

□□□

```
>>>L = [1, 2, 3]
>>>iter(L) is L
False
>>>L.__next__()
AttributeError: 'list' object has no attribute '__next__'

>>>I = iter(L)
>>>I.__next__()
1
>>>next(I)                    # Same as I.__next__()
2
```

□□Python□□□

□□□□□□□□□□□□□Python 2.6□□□□□□□□□X.next()□□□X.__next__()□□□□□□□□□next(X)□□□□□□Python 2.6□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

```
'a'
>>>next(I)
'c'
>>>next(I)
'b'
>>>next(I)
Traceback (most recent call last):
...more omitted...
StopIteration
```

□□□□□□□□□□□□□□□□□keys□□□□□□□□□——for□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>>for key in D:
...     print(key, D[key])
...
a 1
c 3
b 2
```

□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□for□□□□□□□□□shelves□□□Python□□□□□□□□□□□□□□□□□□□□os.popen□□□□□□shell□□□□□□□□□□□□□□□□□□□□□

```
>>>import os
>>>P = os.popen('dir')
>>>P.__next__()
' Volume in drive C is SQ004828V03\n'
>>>P.__next__()
' Volume Serial Number is 08BE-3CD4\n'
>>>next(P)
TypeError: _wrap_close object is not an iterator
```

□□□□Python 2.6□□popen□□□□□□P.next()□□□□Python 3.0□□□□□P.__next__()□□□□□□□next(P)□□□□□□□

```
1
>>>list(range(5))                    # Or use list to collect all results at once
[0, 1, 2, 3, 4]
```

□□□

□□□□□□□

□□□□□□

□□□

```
>>>res = []
>>>for x in L:
...     res.append(x + 10)
...
>>>res
[21, 22, 23, 24, 25]
```

□□□□□□□□□□□□□□□□□□□□□□□□

□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

```
>>>lines = [line.rstrip() for line in open('script1.py')]
>>>lines
['import sys', 'print(sys.path)', 'x = 2', 'print(2 ** 33)']
```

□□□

□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□□20□□□□□

□□□□□□

□□□□□

```
>>>uppers
['IMPORT SYS\n', 'PRINT(SYS.PATH)\n', 'X = 2\n', 'PRINT(2 ** 33)\n']

>>>map(str.upper, open('script1.py'))            # map is an iterable in 3.0
<map object at 0x02660710>

>>>list( map(str.upper, open('script1.py')) )
['IMPORT SYS\n', 'PRINT(SYS.PATH)\n', 'X = 2\n', 'PRINT(2 ** 33)\n']

>>>'y = 2\n' in open('script1.py')
False
>>>'x = 2\n' in open('script1.py')
True
```

□□□□□□□□□□□□□□□□map□□□□□□□□□□□□□□□□

```
>>>sorted(open('script1.py'))
['import sys\n', 'print(2 ** 33)\n', 'print(sys.path)\n', 'x = 2\n']

>>>list(zip(open('script1.py'), open('script1.py')))
[('import sys\n', 'import sys\n'), ('print(sys.path)\n', 'print(sys.path)\n'),
('x = 2\n', 'x = 2\n'), ('print(2 ** 33)\n', 'print(2 ** 33)\n')]

>>>list(enumerate(open('script1.py')))
[(0, 'import sys\n'), (1, 'print(sys.path)\n'), (2, 'x = 2\n'),
(3, 'print(2 ** 33)\n')]

>>>list(filter(bool, open('script1.py')))
['import sys\n', 'print(sys.path)\n', 'x = 2\n', 'print(2 ** 33)\n']

>>>import functools, operator
>>>functools.reduce(operator.add, open('script1.py'))
'import sys\nprint(sys.path)\nx = 2\nprint(2 ** 33)\n'
```

□□□

```
>>>sum([3, 2, 4, 1, 5, 0])          # sum expects numbers only
15
>>>any(['spam', '', 'ni'])
True
>>>all(['spam', '', 'ni'])
False
>>>max([3, 2, 5, 1, 4])
5
>>>min([3, 2, 5, 1, 4])
1
```

□□□□□max□min□□□□□□□□□□——□□□□□□□□□□

```
>>>list(open('script1.py'))
['import sys\n', 'print(sys.path)\n', 'x = 2\n', 'print(2 ** 33)\n']

>>>tuple(open('script1.py'))
('import sys\n', 'print(sys.path)\n', 'x = 2\n', 'print(2 ** 33)\n')

>>>'&&'.join(open('script1.py'))
'import sys\n&&print(sys.path)\n&&x = 2\n&&print(2 ** 33)\n'

>>>a, b, c, d = open('script1.py')


>>>a, d
 ('import sys\n', 'print(2 ** 33)\n')

>>>a, *b = open('script1.py')                    # 3.0 extended form
>>>a, b
('import sys\n', ['print(sys.path)\n', 'x = 2\n', 'print(2 ** 33)\n'])
```

□□□□□□□□□□□dict□□□□□□□□□□zip□□□□□□□□□□□set□□□□□

Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□4□□□5□□□8□□□□□

```
>>>set(open('script1.py'))
{'print(sys.path)\n', 'x = 2\n', 'print(2 ** 33)\n', 'import sys\n'}

>>>{line for line in open('script1.py')}
{'print(sys.path)\n', 'x = 2\n', 'print(2 ** 33)\n', 'import sys\n'}

>>>{ix: line for ix, line in enumerate(open('script1.py'))}
{0: 'import sys\n', 1: 'print(sys.path)\n', 2: 'x = 2\n', 3: 'print(2 ** 33)\n'}
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□"p"□□□□□□□□□□□□□□□□□□□

```
>>>X = (1, 2)
>>>Y = (3, 4)
>>>
>>>list(zip(X, Y))                    # Zip tuples: returns an iterable
[(1, 3), (2, 4)]
>>>
>>>A, B = zip(*zip(X, Y))             # Unzip a zip!
>>>A
(1, 2)
>>>B
(3, 4)
```

Python□□□□□□□□□□□□□□□□range□□□□□□□□□□□□□□□□

Python 3.0□□□□□□□□□□

Python 3.0□□□□□□□□□□□□□□Python 2.X□□□□□□□□□□□□□□

□□□□□□□□□□□□range□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□

□□□□□□□□□□□Python 2.X□□□□□□xrange□□□□□□□□□range□□□□□□□□□□□

map、zip、filter□□□

□range□□□map、zip□□filter□□□□□Python 3.0□□□□□

```
>>>Z = zip((1, 2, 3), (10, 20, 30))      # zip is the same: a one-pass iterator
>>>Z
<zip object at 0x02770EE0>

>>>list(Z)
[(1, 10), (2, 20), (3, 30)]

>>>for pair in Z: print(pair)           # Exhausted after one pass
...

>>>Z = zip((1, 2, 3), (10, 20, 30))
>>>for pair in Z: print(pair)           # Iterator used automatically or manually
...
```

```
(1, 10)
(2, 20)
(3, 30)

>>>Z = zip((1, 2, 3), (10, 20, 30))
>>>next(Z)
(1, 10)
>>>next(Z)
(2, 20)
```

□□□□□□□□□□□□□□□filter□□□□□□□□□□□□□□□□□□□□□□□□□□□True□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□filter□□□□□□□□□□□□□□□□

□□□□□VS□□□□□

□□range□□□□□

```
(1, 10)
>>>next(I1)
(2, 11)
>>>next(I2)                          # I2 is at same spot as I1!
 (3, 12)

>>>M = map(abs, (-1, 0, 1))         # Ditto for map (and filter)
>>>I1 = iter(M); I2 = iter(M)
>>>print(next(I1), next(I1), next(I1))
1 0 1
>>>next(I2)
StopIteration

>>>R = range(3)                      # But range allows many iterators
>>>I1, I2 = iter(R), iter(R)
>>>[next(I1), next(I1), next(I1)]
[0 1 2]
>>>next(I2)
0
```

□□□□□□□□□□29□□□□□□□□□□

□□□□□□□

□□□□□□8□□□□□□□□□□□Python 3.0□□□□□keys□values□items□□□□□□□□□□□

```
>>>K = D.keys()
>>>list(K)                          # Can still force a real list if needed
['a', 'c', 'b']

>>>V = D.values()                   # Ditto for values() and items() views
>>>V
<dict_values object at 0x026D8260>
>>>list(V)
[1, 3, 2]

>>>list(D.items())
[('a', 1), ('c', 3), ('b', 2)]

>>>for (k, v) in D.items(): print(k, v, end=' ')
...
a 1 c 3 b 2
```

□□□Python 3.0□□□

```
>>>D
{'a': 1, 'c': 3, 'b': 2}
>>>for k in sorted(D.keys()): print(k, D[k], end=' ')
...
a 1 b 2 c 3

>>>D
{'a': 1, 'c': 3, 'b': 2}
>>>for k in sorted(D): print(k, D[k], end=' ')   # Best practice key sorting


  ...
  a 1 b 2 c 3
```

□□□□□□□□

□□□□□□□20□□□□□□□

□□□□

□□□□□□□□□□Python□□□□□□□□□□□□□Python□□□□□□□□□□□□□

□□□□

1.for□□□□□□□□□□□□□□□□

2.for□□□□□□□□□□□□□□□□□

3.□□Python□□4□□□□□□

4.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□

第15章 文档

在前面的章节中，我们已经学习了Python的基本语法和数据结构，以及Python的面向对象编程和函数式编程。这些知识点已经足够我们编写一些简单的Python程序了。但是，在实际的软件开发中，我们还需要学习一些其他的知识，比如文档的编写和使用。

文档是软件开发中非常重要的一部分，它可以帮助我们更好地理解和使用Python程序。在Python中，文档通常是以注释的形式存在的。注释是程序中的一些说明性文字，它们不会被Python解释器执行，但是可以帮助我们理解程序的功能和实现方式。在本章中，我们将学习如何编写和使用Python文档。

Python文档

在编写Python程序时，我们可以使用注释来为程序添加说明性文字。注释可以帮助我们更好地理解程序的功能和实现方式，也可以帮助其他人更好地理解我们的程序。

在Python中，我们可以使用两种方式来编写注释。第一种方式是使用单行注释，即在一行中使用#符号来表示注释。第二种方式是使用多行注释，即使用三个单引号或三个双引号来表示注释。多行注释可以跨越多行，通常用于编写较长的说明性文字。

例15-1：使用单行注释和多行注释来为Python程序添加说明性文字。

表15-1：Python文档资源

形式	角色
#注释	文件中的文档
dir函数	对象中可用属性的列表
文档字符串：__doc__	附加在对象上的文件中的文档
PyDoc：help函数	对象的交互帮助
PyDoc：HTML报表	浏览器中的模块文档
标准手册	正式的语言和库的说明
网站资源	在线教程、例子等
出版的书籍	商业参考书籍

#□□

□□□□□□□□□□□□□□□□□□□□Python□□□#□□□□□□□□□#□□□□□□□□□□□□□□□□□□□

dir□□

□□□dir□□□

```
>>>dir([])
['__add__', '__class__', '__contains__', ...more...
'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']

>>>dir('')
['__add__', '__class__', '__contains__', ...more...
'capitalize', 'center', 'count', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'index', 'isalnum', 'isalpha', 'isdecimal',
'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', '
maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
...more names omitted...]
```

□□□□□□□dir□□□□□□□

□□□□□□__doc__

□□#□□□□Python□□□□□

```
"""
Module documentation
Words Go Here
"""

spam = 40

def square(x):
    """
    function documentation
    can we have your liver then?
    """
    return x ** 2           # square

class Employee:
    "class documentation"
    pass

print(square(4))
print(square.__doc__)
```

□□□□□□□□□□□□□□□□□□□□□__doc__□□□□□□□□□□□□□□□□□□□

```
>>>import docstrings
16

   function documentation
   can we have your liver then?

>>>print(docstrings.__doc__)

Module documentation
Words Go Here

>>>print(docstrings.square.__doc__)

   function documentation
   can we have your liver then?

>>>print(docstrings.Employee.__doc__)
   class documentation
```

□□□

□□□□□□□□□□□□HTML□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□

```
>>>print(int.__doc__)
int(x[, base]) -> integer

Convert a string or number to an integer, if possible. A floating
point argument will be truncated towards zero (this does not include a
...more text omitted...

>>>print(map.__doc__)
map(func, *iterables) --> map object

Make an iterator that computes the function using arguments from
each of the iterables. Stops when the shortest iterable is exhausted.
```

□□□

PyDoc□help□□

□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□PyDoc□□□Python□□□□□□□□□□□□□□□□□□□

```
>>>help(sys)
Help on built-in module sys:

NAME
    sys

FILE
    (built-in)

MODULE DOCS
    http://docs.python.org/library/sys

DESCRIPTION
    This module provides access to some objects used or maintained by the
    interpreter and to functions that interact strongly with the interpreter.
    ...more omitted...

FUNCTIONS
    __displayhook__ = displayhook(...)
        displayhook(object) -> None

        Print an object to sys.stdout and also save it in builtins.
        ...more omitted...

DATA
    __stderr__ = <io.TextIOWrapper object at 0x0236E950>
    __stdin__ = <io.TextIOWrapper object at 0x02366550>
    __stdout__ = <io.TextIOWrapper object at 0x02366E30>
    ...more omitted...
```

□□□

```
>>>help(dict)
Help on class dict in module builtins:

class dict(object)
 | dict() -> new empty dictionary.
 | dict(mapping) -> new dictionary initialized from a mapping object's
 ...more omitted...

>>>help(str.replace)
Help on method_descriptor:

replace(...)
    S.replace (old, new[, count]) -> str

    Return a copy of S with all occurrences of substring
    ...more omitted...

>>>help(ord)
Help on built-in function ord in module builtins:

ord(...)
   ord(c) -> integer

   Return the integer ordinal of a one-character string.
```

□□□help□□□

```
>>>import docstrings
>>>help(docstrings.square)
Help on function square in module docstrings:

square(x)
    function documentation
    can we have your liver then?

>>>help(docstrings.Employee)
Help on class Employee in module docstrings:

class Employee(builtins.object)
 | class documentation
 |
 | Data descriptors defined here:
 ...more omitted...

>>>help(docstrings)
Help on module docstrings:

NAME
    docstrings

FILE
    c:\misc\docstrings.py

DESCRIPTION
    Module documentation
    Words Go Here

CLASSES
    builtins.object
        Employee

    class Employee(builtins.object)
     | class documentation
     |
     | Data descriptors defined here:
     ...more omitted...

FUNCTIONS
    square(x)
        function documentation
           can we have your liver then?

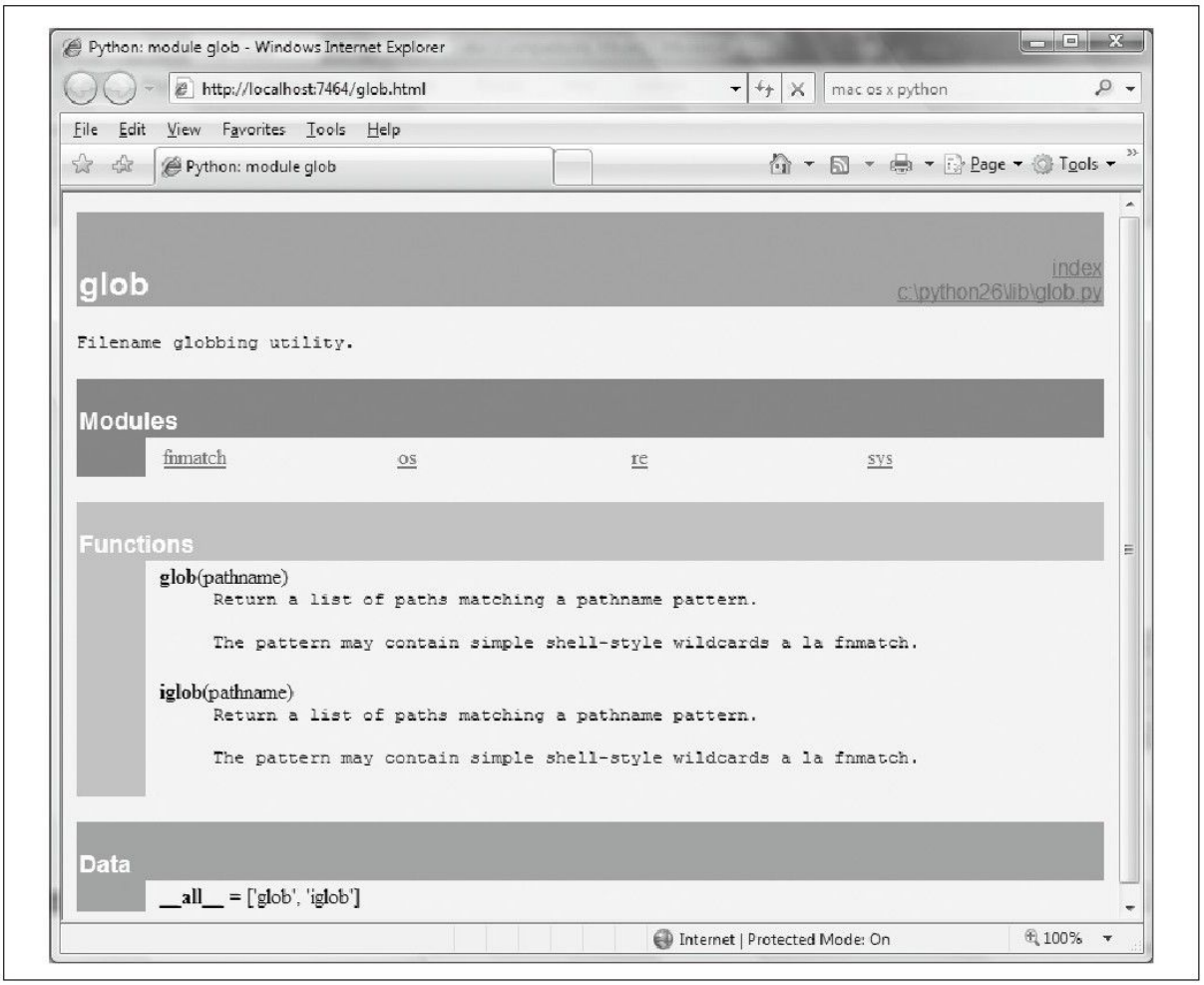
    DATA
       spam = 40
```

PyDoc□HTML□□

□□□□□□□□□□help□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□PyDoc□□□GUI□□□□□□□□□□□□Python/Tkinter□□□□□□□□□□□□□□HTML□□□□□□□□□□□□□□□□□□□

□□

□□□□□□□□□□□□□□□□□"go to selected"□Python□□□□□□□□□□□□□□□□□HTML□□□□□□□□□15-2□□□□□□□□glob□□□PyDoc□□□



□ 15-2 □□□□15-1GUI□□□□□□□□□□□"go to selected"□□□□□□□□□□HTML□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□"Module"□□□□□□□□□□□□□□□□□□□

□□help□□□□□GUI□□□□□□□□□□□□□□□□□□□□15-3□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□PyDoc□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□Python□□□HTML□□□□□□□□□□□Windows□□□□Python□□□□□□□□□“□□”□□□Python□□□□□□□□□□□□□IDLE□"Help"□□□□□□□□□□□□□□http://www.python.org□□□□□□□□□□□□□□□□□□□□□□□□□□□□Documentation□□□□□Windows□□□□□□□□□□□□□□□□□□□□□□□□Python.org□□□□□□□□□□□□□□□□

□□□□Windows□□□□□□□□□□□15-4□□□□□□□□□□□□□□□□□□□□□□□□□□□□□"Library Reference"□□□□□□□□□□□□□□□□□□□



□ 15-4 Python□□□□□□□□www.python.org□□□□□□□IDLE □"Help"□□□□□□□□Windows□“□□”□□□□□□□□□Windows□□□□□□□□ □□□□□□□□□□□□□□□□□□□□Library□□□□□□□□□□□□□□□□□□

□□□□

□□□□Python□□□□□□□□□□□http://www.python.org□□□□□□□□□
Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Documentation□□□□□□□□
□□□□□"Beginners Guide to Python"□□□□□□□□□□□□□□Python□□□

□□□□□□Web□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

·□□□Python□□C□□□□C/C++□□□□□□□□□if□while□□□□□□□□□□□□□□□□□□□if(X==1):□□□□□□□□□□□□□□□□□□□

sort□□□□None□□□□□□□□□□□□□□□□□None□□□□□□□□□□□□□□□□□

□□□□

□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□#□□□□

2.□□3□□□□□□□□□□□□□

3.□□□□□□□□□□□□□□□□□

4.□□□□□□□□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□Python□□□

□□□□

1.□□□

□□□□ □□□

□□□

□□□□□□□□IDLE□□□□□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□

□□□□ □□

□16□ □□□□

□□□□□□□□□□Python□□□□□□□□□□

表 16-1：函数相关的语句和表达式

语句	例子
Calls	`myfunc("spam", "eggs", meat=ham)`
def, return	`def adder(a, b=1, *c):` `    return a+b+c[0]`

表 16-1：函数相关的语句和表达式（续）

语句	例子
global	`def changer():` `    global x; x = 'new'`
nonlocal	`def changer():` `    nonlocal x; x = 'new'`
yield	`def squares(x):` `    for i in range(x): yield i ** 2`
lambda	`Funcs = [lambda x: x**2, lambda x: x*3]`

□□□□□□

□□□□□

□□□□□□□□□□□□Python□□□□□□□□□□□□

□□□□

□□□

□□□

·return□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
def <name>(arg1, arg2,... argN):
    ...

return <value>
```

Python□return□□□

def□□□□□□□□

Python□def□□□□□□□□

□□□

□□□□□□□□□□□□

□□□□□□□□□□□

□□

□def□□□

□□□□□□□□□□□□□□□Monty Python□□□□□□□□□□□□□□□

Python□□□□

□□□□□□□□□□□□times□□□□□□□x*y□□□□□□□□□x□y□□□□□□□□□□□□

□□□Python□□□□□□□□□□C++□Java□□□□□□□□□□□□Python□□□□□□□□□□□□

□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□

□□□

□□

□□□

□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□

□□□□Python□□□□□□□intersect□□□□□□□

[1]□□□□□□□file.readlines()□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□intersect□□□□res□□□□Python□□□□□□□□——□□□□□□□□def□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□——□□□□def□return□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□

2.□□□□Python□□□□□□□

3.□□□□□□□□return□□□□□□□□□□□□

4.□□□□□□□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□Python□□□

第17章　　

第16章介绍了如何使用Python来处理数据，包括Python的数据结构、Python的数据处理方法、Python的数据可视化等。这些内容都是Python数据处理的基础，掌握这些内容对于使用Python进行数据处理是非常重要的。

Python的应用

在实际的数据处理过程中，我们经常会遇到一些复杂的数据处理问题，这时候Python的强大功能就可以发挥出来了。Python的应用非常广泛，可以用于数据处理、数据分析、数据可视化、机器学习等多个领域。在这些领域中，Python都有着非常强大的功能和丰富的库支持。

在数据处理方面，Python的应用非常广泛。Python的数据处理库非常丰富，包括NumPy、Pandas、Matplotlib等。这些库可以帮助我们进行数据的读取、处理、分析和可视化。Python的数据处理库非常强大，可以处理各种类型的数据，包括结构化数据、非结构化数据、时间序列数据等。

在数据分析方面，Python的应用也非常广泛。Python的数据分析库非常丰富，包括SciPy、Statsmodels、Scikit-learn等。这些库可以帮助我们进行数据的统计分析、机器学习、深度学习等。Python的数据分析库非常强大，可以处理各种类型的数据，包括结构化数据、非结构化数据、时间序列数据等。

在数据可视化方面，Python的应用也非常广泛。Python的数据可视化库非常丰富，包括Matplotlib、Seaborn、Plotly等。这些库可以帮助我们进行数据的可视化，包括折线图、柱状图、散点图、热力图等。Python的数据可视化库非常强大，可以处理各种类型的数据，包括结构化数据、非结构化数据、时间序列数据等。

在机器学习方面，Python的应用也非常广泛。Python的机器学习库非常丰富，包括Scikit-learn、TensorFlow、Keras等。这些库可以帮助我们进行机器学习的模型训练、模型评估、模型预测等。Python的机器学习库非常强大，可以处理各种类型的数据，包括结构化数据、非结构化数据、时间序列数据等。

·□□□def□□□□□□□□□□def□□□□□□□□□□

```
X = 99

def func():
    X = 88
```

□□□□□□□□□□X□□□□□□□□□□

·□□□

□□□

□□□□□□LEGB□□

□□□□□

·□□□

[1]□□□□□□□□□□□□□□□□□□□□□□LGB□□□“□□def□”□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□

□□□

□□□□□

□□□□

```
>>>import builtins                    # The hard way
>>>builtins.zip
<class 'zip'>
```

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□X□□□□□□□□□□□□□□□□□□□□X□□□□□□□□□□□□□□□□□□□

True□False□□□□□□□□□□□□□□None□□□□□□□Python 2.6□□□□IDLE□
□□□□□□□□□□□□□□

global□□

global□□□Python□□□

```
def func():
    global X
    X = 99                          # Global X: outside def

func()
print(X)                            # Prints 99
```

□□□□□□□□□□□□□□global□□□□□□□def□□□X□□□□□□def□□□X□□□□□□□□□□□□□□□□□□□□□□□□global□□□□□□□

```
y, z = 1, 2                         # Global variables in module
def all_global():
    global x                        # Declare globals assigned
    x = y + z                       # No need to declare y, z: LEGB rule
```

□□□x□y□z□□all_global□□□□□□□□□□y□z□□□□□□□□□□□□□□□□□□□□□□□□□□□□x□□□□□□□□□□□□□global□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□global□□□□□□□□□□□def□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□

□□□□□□□□

X□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
>>>import thismod
>>>thismod.test()
99
102
>>>thismod.var
102
```

□□□

□□□□□□□□□

□□□□□□□□□□Python□□□□□□□□□□

□□□□□□□

□□□

```
X = 99                          # Global scope name: not used

def f1():
    X = 88                      # Enclosing def local
    def f2():
        print(X)                # Reference made in nested def
    f2()

f1()                            # Prints 88: enclosing def local
```

□□□□□□□□□□□Python□□□def□□□□□□□□□□□□□□□□□□□

```
def f1():
    X = 88
   def f2():
       print(X)                    # Remembers X in enclosing def scope
   return f2                       # Return f2 but don't call it

action = f1()                      # Make, return function
action()                           # Call it now: prints 88
```

□□□□□□□□□□□□□□f2□□□□□□□□□□□□□□□f1□□□□□□□□f2□□□□f1□□□□□□□□x□□□f1□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□closure□□□□□□□□——□□□□□□□□□□□□□□□□□□□

```
>>>f = maker(2)                    # Pass 2 to N
>>>f
<function action at 0x014720B0>
```

□□□

□□□

□□□□□f2□x=x□□□□□x□□□□□□□□□□□□x□□□□□□□□□□x□Python□□□□□def□□□□□□□□□□□□□□□□f1□□x□□□□□□□□□□□□f1□x□□□□□□□□□88□□

□□□□□□□□□□□□□□□□

□□□□def□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□lambda□□def□□□□□□□□□□□□□□□□□□□

□□□

```
>>>def f1():
...     x = 99
...     def f2():
...         def f3():
...             print(x)                    # Found in f1's local scope!
...         f3()
...         f2()
...
>>>     f1()
99
```

Python□□□□□□□□□def□□□□□□□□□

nonlocal□□

□□□

□□□

·global□□□

nonlocal□□

□□□□□□Python 3.0□□□□□□□□□def□□□□□□□□Python 2.6□□□□□□□□□□□□□□□tester□□□□□□□nested□□□□□□□nested□□state□□□□□□□□□□□□□□□□□□□□tester□□□□□□□

```
C:\\misc>c:\python30\python

>>>def tester(start):
...     state = start                   # Referencing nonlocals works normally
...     def nested(label):
...         print(label, state)         # Remembers state in enclosing scope
...     return nested
...
>>>F = tester(0)
>>>F('spam')
spam 0
>>>F('ham')
ham 0
```

□□□□□□□□□□□□□□□□def□□□□□□□□□□□□Python 2.6□□□□□□

```
>>>def tester(start):
...     state = start
...     def nested(label):
...         print(label, state)
...         state += 1                  # Cannot change by default (or in 2.6)
...      return nested
...
>>>F = tester(0)
>>>F('spam')
UnboundLocalError: local variable 'state' referenced before assignment
```

□□nonlocal□□□□

□□□□Python 3.0□□□□□□□□nested□□tester□□□□□state□□□□□nonlocal□□□□□□□□nested□□□□□□□□□□□□□□□□□□□□F□□□□□nested□□□□tester□□□□□□□□□□□□□□□□□□

```
>>>def tester(start):
...     state = start                  # Each call gets its own state
...     def nested(label):
...         nonlocal state             # Remembers state in enclosing scope
...         print(label, state)
...         state += 1                 # Allowed to change it if nonlocal
...     return nested
...
>>>F = tester(0)
>>>F('spam')                           # Increments state on each call
spam 0
>>>F('ham')
ham 1
>>>F('eggs')
eggs 2
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□tester□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□global□□□□□□□□□□□nonlocal□□□□nonlocal□□□□□□□□□□□□□def□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

```
>>>spam = 99
>>>def tester():
...     def nested():
...         nonlocal spam                # Must be in a def, not the module!
...         print('Current=', spam)
...         spam += 1
...      return nested
...
SyntaxError: no binding for nonlocal 'spam' found
```

□□□□□□Python□□□□□□

□□□□□nonlocal

□□□□

```
>>>def tester(start):
...     global state                    # Move it out to the module to change it
...     state = start                   # global allows changes in module scope
...     def nested(label):
...         global state
...         print(label, state)
...         state += 1
...     return nested
...
>>>F = tester(0)
>>>F('spam')                           # Each call increments shared global state
spam 0
>>>F('eggs')
eggs 1
```

□□□

Python 2.6□□□

□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□

□□□nonlocal□□□

□□□□

□□□

□□□□

1.□□□□□□□□□□□□□□□□□□

```
>>>X = 'Spam'
>>>def func():
...    print(X)
...
>>>func()
```

2.□□□□□□□□□□□□□□□□□□

```
>>>X = 'Spam'
>>>def func():
...    X = 'NI!'
...
>>>func()
>>>print(X)
```

3.□□□□□□□□□□□□□□□□□□□□

```
>>>X = 'Spam'
>>>def func():
...     X = 'NI'
...     print(X)
...
>>>func()
>>>print(X)
```

4.□□□□□□□□□□□□□□□□

```
>>>X = 'Spam'
>>>def func():
...     global X
...     X = 'NI'
...
>>>func()
>>>print(X)
```

5.□□□□□□□□□□□□□□□□

```
>>>X = 'Spam'
>>>def func():
...     X = 'NI'
...     def nested():
...         print(X)
...     nested()
...
>>>func()
>>>X
```

6.□□□□□Python 3.0□□□□□□□□□□□□

```
>>>def func():
...     X = 'NI'
...     def nested():
...         nonlocal X
...         X = 'Spam'
...     nested()
...     print(X)
...
>>>func()
```

7.□□□□□□□Python□□□□□□□□□□□□□

□□□□

1.□□□□□□'Spam'□□□□□□

□□□

第18章　函数

第17章介绍了Python的基本数据类型——整数、浮点数、字符串、列表、元组、字典和集合。这些数据类型是Python语言的基础，本章将介绍Python的函数。函数是一段可以重复使用的代码，可以接收argument（也称为parameter）作为输入，并返回一个结果。Python的函数可以是内置的，也可以是用户自己定义的。

本章内容

本章将介绍函数的基本概念和使用方法，包括函数的定义、调用、参数传递、返回值、作用域、递归等。

·函数的基本概念，包括函数的定义、调用、参数传递、返回值等[本章将介绍函数的基本概念，包括函数的定义、调用、参数传递、返回值等]。本章将介绍Python的函数的基本概念，包括函数的定义、调用、参数传递、返回值等。

·函数的参数传递，包括位置参数、关键字参数、默认参数、可变参数等。本章将介绍函数的参数传递，包括位置参数、关键字参数、默认参数、可变参数等。

·函数的作用域，包括全局变量、局部变量、闭包等。本章将介绍函数的作用域，包括全局变量、局部变量、闭包等。

□□□□□□□□□□□□□□6□□□□□□□□□□□□□□

□□□□□□□□□□□f(b)□□□□□□□□□□□a□□□□□88□□□□a□□□□□□□□□□□

·b□□□

```
>>>X = 1
>>>a = X                    # They share the same object
>>>a = 2                    # Resets 'a' only, 'X' is still 1
>>>print(X)
1
```

□□□

□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□bug——□□□□□□□□Python□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
L = [1, 2]
changer(X, tuple(L))                    # Pass a tuple, so changes are errors
```

□□□□□□□□□□tuple□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□return□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□return□□□□□□□□□□□□□□□□□□□

```
def f((a, (b, c))):
```

□□□□□□□□□□□□□□□□□□□□□□□□f((1,(2,3)))□□□a□b□c□□□1□2□3□□□□□□□□□□□□□□□□□f(T)□□□□□□□□□□□□□□□□def□□□Python 3.0□□□□□□□□□□□□□□□□□□□□

```
def f(T): (a, (b, c)) = T
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□Python 2.6□□□□□□□□□□□Python 2.X□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□Python□□□□□□□□□□□□

□□□

□□□□

□18-1□□□□□□□□□□□□□□□□□□□□

表18-1：函数参数匹配表

语法	位置	解释
func(value)	调用者	常规参数：通过位置进行匹配
func(name=value)	调用者	关键字参数：通过变量名匹配
func(*sequence)	调用者	以name传递所有的对象，并作为独立的基于位置的参数
func(**dict)	调用者	以name成对的传递所有的关键字/值，并作为独立的关键字参数
def func(name)	函数	常规参数: 通过位置或变量名进行匹配

表18-1：函数参数匹配表（续）

语法	位置	解释
def func(name=value)	函数	默认参数值，如果没有在调用中传递的话
def func(*name)	函数	匹配并收集（在元组中）所有包含位置的参数
def func(**name)	函数	匹配并收集（在字典中）所有包含位置的参数
def func(*args, name) def func(*, name=value)	函数	参数必须在调用中按照关键字传递 （Python 3.0）

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□□□□□4□□□□□□□□□□□□□□□□□□□□□□□□□□□□name=value□□□□□Python□□□□□□□□□□□□□□□□□□□

**dict□□□

□□

□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□value□□□□□□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□

□□□□□□□□□□□□□

□□□□□□□□□c=3□□□□□3□□□□□c□□□□□□□Python□□□□□□□□c□□□□□□□□□□□c□□□□□□□3□□□□□□□□□□□

```
>>>def f(a, b=2, c=3): print(a, b, c)
...
```

□□□□□□□□□□□□□□□□□a□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□name=value□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□*□**□□□□□□□□□□

□□□□□□□□□□□□□□□□

```
>>>def f(**args): print(args)
...
>>>f()
{}
>>>f(a=1, b=2)
{'a': 1, 'b': 2}
```

□□□□□□□□□□□□□□□□□□*□□□□**□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□a□2□3□□□pargs□□□□□□□x□y□□kargs□□□□□□□□

```
>>>def f(a, *pargs, **kargs): print(a, pargs, kargs)
...
>>>f(1, 2, 3, x=1, y=2)
1 (2, 3) {'y': 2, 'x': 1}
```

□□□□□□□□□□□□□□□□□□□

```
>>>def func(a, b, c, d): print(a, b, c, d)
...
>>>args = (1, 2)
>>>args += (3, 4)
>>>func(*args)
1 2 3 4
```

□□□□□□□□□□□□**□□□/□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>>args = {'a': 1, 'b': 2, 'c': 3}
>>>args['d'] = 4
>>>func(**args)
1 2 3 4
```

□□□□□□□□□□□□□□□□□□□

```
>>>f(1, *(2,), c=3, **{'d':4})
1 2 3 4
```

□□□

```
if <test>:
    action, args = func1, (1,)          # Call func1 with 1 arg in this case
else:
    action, args = func2, (1, 2, 3)     # Call func2 with 3 args here
...
action(*args)                           # Dispatch generically
```

□□□

```
calling: func
10
```

□□□

□Python 2.6□□□□□□□□apply□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□

```
>>>pargs = (1, 2)
>>>kargs = {'a':3, 'b':4}

>>>apply(echo, pargs, kargs)
(1, 2) {'a': 3, 'b': 4}
```

```
>>>echo(*pargs, **kargs)
(1, 2) {'a': 3, 'b': 4}
```

□□□□□□□□□apply□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□def□□□*pargs□**kargs□□□□□□□□□□□□□□□□□

Python 3.0 Keyword-Only□□

Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□keyword-only□□——□□□□□□□□□□□□□□□□□□□

```
>>>def kwonly(a, *, b, c):
...     print(a, b, c)
...
>>>kwonly(1, c=3, b=2)
1 2 3
>>>kwonly(c=3, b=2, a=1)
1 2 3
>>>kwonly(1, 2, 3)
```

```
TypeError: kwonly() takes exactly 1 positional argument (3 given)
>>>kwonly(1)
TypeError: kwonly() needs keyword-only argument b
```

□□□□□keyword-only□□□□□□□□□□□□□□□□□□□□□□□□□*□□□□□□□□□□□□a□□□□□□□□□□□□□□□b□c□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>>def kwonly(a, *, b='spam', c='ham'):
...     print(a, b, c)
...
>>>kwonly(1)
1 spam ham
>>>kwonly(1, c=3)
1 spam 3
>>>kwonly(a=1)
1 spam ham
>>>kwonly(c=3, b=2, a=1)
1 2 3
>>>kwonly(1, 2)
TypeError: kwonly() takes exactly 1 positional argument (2 given)
```

□□□□□□□□□□□keyword-only□□□□□□□□□□□□□□□□□□□□□□keyword-only□□□□□□□□□□□□□□□□□keyword-only□□□

```
>>>def kwonly(a, *, b, c='spam'):
...     print(a, b, c)
...
>>>kwonly(1, b='eggs')
1 eggs spam
>>>kwonly(1, c='eggs')
TypeError: kwonly() needs keyword-only argument b
>>>kwonly(1, 2)
TypeError: kwonly() takes exactly 1 positional argument (2 given)

>>>def kwonly(a, *, b=1, c, d=2):
...     print(a, b, c, d)
...
>>>kwonly(3, c=4)
3 1 4 2
>>>kwonly(3, c=4, b=5)
3 5 4 2
>>>kwonly(3)
TypeError: kwonly() needs keyword-only argument c
>>>kwonly(1, 2, 3)
TypeError: kwonly() takes exactly 1 positional argument (3 given)
```

□□□□

□□□□□keyword-only□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□**args□□□□□□□□□□□□□□□□□**□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>>def kwonly(a, **pargs, b, c):
SyntaxError: invalid syntax
>>>def kwonly(a, **, b, c):
SyntaxError: invalid syntax
```

□□□□□□□□□□□□□□□keyword-only□□□□□□□□**args□□□□□□□□□□□□□□□*args□□□□□□□□□□□□□□□

```
>>>def f(a, *b, c=6, **d): print(a, b, c, d)              # KW-only between * and **
...
>>>f(1, *(2, 3), **dict(x=4, y=5))                       # Unpack args at call
1 (2, 3) 6 {'y': 5, 'x': 4}

>>>f(1, *(2, 3), **dict(x=4, y=5), c=7)                  # Keywords before **args!
SyntaxError: invalid syntax

>>>f(1, *(2, 3), c=7, **dict(x=4, y=5))                  # Override default
1 (2, 3) 7 {'y': 5, 'x': 4}

>>>f(1, c=7, *(2, 3), **dict(x=4, y=5))                  # After or before *
1 (2, 3) 7 {'y': 5, 'x': 4}

>>>f(1, *(2, 3), **dict(x=4, y=5, c=7))                  # Keyword-only in **
1 (2, 3) 7 {'y': 5, 'x': 4}
```

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
def process(*args, notify=False): ...
```

□□□□□“□□Python 3.0 print□□”□□□□□□□□□□□□□□□□□□□

min□□

□□□

□□

□□□

```
def min1(*args):
    res = args[0]
    for arg in args[1:]:
        if arg < res:
            res = arg
    return res

def min2(first, *rest):
    for arg in rest:
        if arg < first:
            first = arg
    return first

def min3(*args):
    tmp = list(args)                    # Or, in Python 2.4+: return sorted(args)[0]
    tmp.sort()
    return tmp[0]

print(min1(3,4,1,2))
print(min2("bb", "aa"))
```

```
print(min3([2,2], [1,1], [3,3]))
```

□□□

□□□

□□□

□□□

□□

□□□

□□□□□□□□□□□□set□□

□□□□□□□□□□□

intersect□□□

□□Python 3.0 print□□

□□□Python 2.6 □□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□*args□□□□□□□□□ **args□□□□□□□□□□□□□□Python 3.0 print□□□□□□□□□□□□□

□□□□□□□11□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□ □□□□□□□□□□Python 3.0□print□□□

```
from __future__ import print_function
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□print30.py□□□□□□□□□□□□□□□□□□□□□□

```
"""
Emulate most of the 3.0 print function for use in 2.X
call signature: print30(*args, sep=' ', end='\n', file=None)
```

```
"""
import sys

def print30(*args, **kargs):
    sep = kargs.get('sep', ' ')                    # Keyword arg defaults
    end = kargs.get('end', '\n')
    file = kargs.get('file', sys.stdout)
    output = ''
    first = True
    for arg in args:
        output += ('' if first else sep) + str(arg)
        first = False
    file.write(output + end)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0 print□□□□□□□□□□□□□□□□□□□□testprint30.py□□□□□□□□□□□□□"print30"□□□"print"□Python 2.6□□□□□□□□

```
from print30 import print30
print30(1, 2, 3)
print30(1, 2, 3, sep='')                           # Suppress separator
print30(1, 2, 3, sep='...')
print30(1, [2], (3,), sep='...')                   # Various object types

print30(4, 5, 6, sep='', end='')                   # Suppress newline
print30(7, 8, 9)
print30()                                          # Add newline (or blank line)

import sys
print30(1, 2, 3, sep='??', end='.\n', file=sys.stderr)    # Redirect to file
```

□Python 2.6□□□□□□□□□□□□□□□□Python 3.0□print□□□□□□□□□

```
C:\misc>c:\python26\python testprint30.py
1 2 3
123
1...2...3
1...[2]...(3,)
4567 8 9
1??2??3.
```

□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□Python□□□□□□□□□□□□□□□

□□Keyword-Only□□

□□□□□□□□□□□□□□□□□□Python 3.0 keyword-only□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□keyword-only□□□□□□□□□□□□□□□□□□□

```
>>>print30(99, name='bob')
TypeError: extra keywords: {'name': 'bob'}
```

□□□□□□□□Python 2.6□□□□□□□□□keyword-only□□□□□□□□□4□□□□□□□□□

□□□

□□□□

□□□

□□□□

1.□□□□□□□□□□□□□□□□□

```
>>>def func(a, b=4, c=5):
...    print(a, b, c)
...
>>>func(1, 2)
```

2.□□□□□□□□□□□□□□□□□

```
>>>def func(a, b, c=5):
...    print(a, b, c)
...
>>>func(1, c=3, b=2)
```

3.□□□□□□□□□□□□□□□□□

```
>>>def func(a, *pargs):
...    print(a, pargs)
...
>>>func(1, 2, 3)
```

4.□□□□□□□□□□□□□□□□

```
>>>def func(a, **kargs):
...     print(a, kargs)
...
>>>func(a=1, c=3, b=2)
```

5.□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>>def func(a, b, c=3, d=4): print(a, b, c, d)
...
>>>func(1, *(5,6))
```

6.□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□“125”□□□1□2□□□□□□□□a□b□□□c□□□□□□□□□□□□□□5□

2.□□□□□□“123”□1□□□□□□□a□2□3□□□□□□□b□c□□□□□□□□□□□□□□

第19章　□□□□□□□

□□□□□□□

·□□□

□□□□□□□□□□□□□□□self.name='bob'□□□□□□□□□□□□□□□

□□□□

□□17□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□

□□□□□□□□□□□□□Python□□□if/else□□□□□□12□□□□□□□□□□□□□

□□□

□□□□VS□□

□□□

```
>>>sum = 0
>>>for x in L: sum += x
...
>>>sum
15
```

□□□

□□□□□□

□□□□□

□□□

□□□□□□□□□□

Python□□□□□□□□□

```
>>>def echo(message):              # Name echo assigned to function object
...     print(message)
...
>>>echo('Direct call')            # Call object through original name
Direct call

>>>x = echo                       # Now x references the function too
>>>x('Indirect call!')            # Call object through name by adding ()
Indirect call!
```

□□□

□□□□□□□

```
>>>def make(label):                    # Make a function but don't call it
...     def echo(message):
...         print(label + ':' + message)
...     return echo
...
>>>F = make('Spam')                    # Label in enclosing scope is retained
>>>F('Ham!')                           # Call the function that make returned
Spam:Ham!
>>>F('Eggs!')
Spam:Eggs!
```

Python□□□□□□□

□□□□

□□□

```
>>>func.__code__
<code object func at 0x0257C9B0, file "<stdin>", line 1>

>>>dir(func.__code__)
['__class__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__',
...more omitted...
'co_argcount', 'co_cellvars', 'co_code', 'co_consts', 'co_filename',
'co_firstlineno', 'co_flags', 'co_freevars', 'co_kwonlyargcount', 'co_lnotab',
'co_name', 'co_names', 'co_nlocals', 'co_stacksize', 'co_varnames']

>>>func.__code__.co_varnames
('a', 'b')
>>>func.__code__.co_argcount
1
```

□□□

□□□□

□□17□□□□□□□□□□□□□□□□□□□

Python 3.0□□□□□□

□Python 3.0□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>>def func(a: 'spam', b: (1, 10), c: float) -> int:
...     return a + b + c
...
>>>func(1, 2, 3)
6
```

□□□

```
>>>func.__annotations__
{'a': 'spam', 'c': 99}

>>>for arg in func.__annotations__:
...   print(arg, '=>', func.__annotations__[arg])
...
a => spam
c => 99
```

□□□

□□□Python 3.0□□□

□□□□□lambda

□□def□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□LISP□□□□□□□□□□□□□□□□□lambda□[1]□□def□□□□□□□□□□□□□□□□□□□

·lambda□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□lambda□□□□□□□□□□□□□□□□□□□

```
>>>x = (lambda a="fee", b="fie", c="foe": a + b + c)
>>>x("wee")
'weefiefoe'
```

□lambda□□□□□□□□□def□□□□□□□□□□□□□□□□□□□□□□□□□□□lambda□□□□□□□□□□□□□□□□□□□

□□□□□lambda

□□□□□lambda□□□□□

```
def f1(x): return x ** 2
def f2(x): return x ** 3                    # Define named functions
def f3(x): return x ** 4

L = [f1, f2, f3]                            # Reference by name

for f in L:
    print(f(2))                             # Prints 4, 8, 16

print(L[0](3))                              # Prints 9
```

□□□□□□□□□Python□□□□□□□□□

```
>>>def f1(): return 2 + 2
...
>>>def f2(): return 2 * 4


...
>>>def f3(): return 2 ** 6
...
>>>key = 'one'
>>>{'already': f1, 'got': f2, 'one': f3}[key]()
64
```

□□□□□□□□□□□□□□□□def□□□□□□□□□□□□□□□□

□□□□□□□Python□□□□□□□□

□□lambda□□□□□□□□

```
>>>lower = (lambda x, y: x if x < y else y)
>>>lower('bb', 'aa')
'aa'
>>>lower('aa', 'bb')
'aa'
```

□□□□□□□□lamdba□□□□□□□□□□□□□map□□□□□□□□□□□□□

□□lambda□□□□

lambda□□□□□□□□□□□□□□□□□17□□□□LEGB□□□□E□□□□□□□□□□□□□□□□□□□□□□lambda□□□def□□□□□□□□□□□□□□□□□□□

□□□□□lambda□□□□□

□□□□□□□□□□□□□□lambda□□□□□□□□□□□□□□

□□□□□□□□□□map

□□□□□□□□□

3.0□□□

```
>>>pow(3, 4)                                  # 3**4
81
>>>list(map(pow, [1, 2, 3], [2, 3, 4]))      # 1**2, 2**3, 3**4
[1, 8, 81]
```

□□□□□□□map□□□□N□□□□□□□N□□□□□□pow□□□□□□□

□□□□□□□□filter□reduce

□Python□□□□□□map□□□□□□□

reduce□Python 2.6□□□□□□□□□□□□□□□□□□Python 3.0□□□□□functools□□□□□□□□□□□□□□□□□□

```
>>>def myreduce(function, sequence):
...     tally = sequence[0]
...     for next in sequence[1:]:
...         tally = function(tally, next)
...     return tally
...
>>>myreduce((lambda x, y: x + y), [1, 2, 3, 4, 5])
15
>>>myreduce((lambda x, y: x * y), [1, 2, 3, 4, 5])
120
```

□□□□□reduce□□□□□

□□□□

□□□

□□□□

1.lambda□□□□def□□□□□□□□

2.□□lambda□□□□□□□

3.□□□□□map□filter□reduce□

4.□□□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□

6.□□□□□□□□□□□□□□□□

□□□□

1.lambda□def□□□□□□□□□□□□□□□□□□□□□□□□□□□□lambda□□□□□□□□□□□□□□□□□□□def□□□□□□□□□□□□□□□□lambda□□□□□□□□□□□def□□□□□□□□□□□□□□□□□□□

5.□□□

第20章　□□□□□□□□□□

□□□□□□□□□□□□□□□□□14□□□□□□□□□□□□□□□□□

□□□□□map

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——map□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□

□□□□□□□□

```
>>>[x ** 2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
```

□□□□□□□□□0□9□□□□□□□□□□□□□if□□□□□□□□□for□□□□□□□□□□□□□

```
>>>res = []
>>>for x in [0, 1, 2]:
...     for y in [100, 200, 300]:
...         res.append(x + y)
...
>>>res
[100, 200, 300, 101, 201, 301, 102, 202, 302]
```

□□□

```
>>>res = []
>>>for x in range(5):
...     if x % 2 == 0:
...       for y in range(5):
...           if y % 2 == 1:
...               res.append((x, y))
...
>>>res
[(0, 1), (0, 3), (2, 1), (2, 3), (4, 1), (4, 3)]
```

□□□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>>[row[1] for row in M]
[2, 5, 8]

>>>[M[row][1] for row in (0, 1, 2)]
```

[2, 5, 8]

□□□

```
>>>res = []
>>>for row in range(3):
...     tmp = []
...     for col in range(3):
...         tmp.append(M[row][col] * N[row][col])
...     res.append(tmp)
...
>>>res
[[2, 4, 6], [12, 15, 18], [28, 32, 36]]
```

□□□

□□□□□□

□□□□□

```
>>>open('myfile').readlines()
['aaa\n', 'bbb\n', 'ccc\n']
```

□□□□□□□□□□□□□□□□□□□□□□map□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□map□□□Python 3.0□□□□□□□□□□□□□□□□□□□□list□□□□□□□□□□□□□□□□□□□□□□

```
>>>[line.rstrip() for line in open('myfile').readlines()]
['aaa', 'bbb', 'ccc']

>>>[line.rstrip() for line in open('myfile')]
['aaa', 'bbb', 'ccc']

>>>list(map((lambda line: line.rstrip()), open('myfile')))
['aaa', 'bbb', 'ccc']
```

□□□□□□□□□□□□□□□□□□□

```
>>>[age for (name, age, job) in listoftuple]
[35, 40]

>>>list(map((lambda row: row[1]), listoftuple))
[35, 40]
```

□□□

□□□□□□□□□□

□□Python□□□□□□□□□□□□——□□□□□□□□□□□□

□□□□

□□□

□□□□□□StopIteration□□□□□□□□□□□□□□□□□□□□□□□□□□__next__□□□□□□□□□□□□□□□yield□□□□□□□□□□StopIteration□□□

□□□□□□□□□□□□□□□□□□□□yield□□□def□□□□□□□□□□□□□□□□□□□

```
>>>for i in gensquares(5):                # Resume the function
...     print(i, end=' : ')                # Print last yielded value
...
0 : 1 : 4 : 9 : 16 :
>>>
```

□□□

```
>>>next(x)              # Same as x.__next__() in 3.0
0
>>>next(x)              # Use x.next() or next() in 2.6
1
>>>next(x)
4
>>>next(x)
9
>>>next(x)
Traceback (most recent call last):
...more text omitted...
StopIteration
```

□□□□□□14□□□□□□□□for□□□□□□□□

```
>>>for x in [n ** 2 for n in range(5)]:
...     print(x, end=' : ')
...
0 : 1 : 4 : 9 : 16 :

>>>for x in map((lambda n: n ** 2), range(5)):
...     print(x, end=' : ')
...
0 : 1 : 4 : 9 : 16 :
```

□□□

□□□□□□□□□□□□□□□□□□□□□□□□G.send(value)□□□□□□□□□G□□□□□□□□□□□□□□□□□yield□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□G.__next__()□□□□□□□□□□□next(G)□□yield□□None□□□□

```
>>>def gen():
...     for i in range(10):
...         X = yield i
...         print(X)
...
>>>G = gen()
>>>next(G)                    # Must call next() first, to start generator
0
>>>G.send(77)                 # Advance, and send value to yield expression
77
1
>>>G.send(88)
88
2
>>>next(G)                    # next() and X.__next__() send None
None
3
```

□□□□send□□□□□□□□□□□□□□□□□□□

[1]□□□□□□□□□□□□□□□□□“□□□”□□□□□□□□□□□□□□□□□□□□yield□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□

□□□□□□Python□□□□□□□□□□□□□□□□

```
>>>G = (x ** 2 for x in range(4))
>>>next(G)
0
>>>next(G)
1
>>>next(G)
4
>>>next(G)
9
>>>next(G)

Traceback (most recent call last):
...more text omitted...
StopIteration
```

□□□□□□□□□□□next□□□□□□□□□□□□□□□□for□□□□□□□□

```
>>>for num in (x ** 2 for x in range(4)):
...    print('%s, %s' % (num, num / 2.0))
...
0, 0.0
1, 0.5
4, 2.0
9, 4.5
```

□□□□□□□□□□□□□□□□□□□□□sum□map□sorted□□□□□□□□□□14□□□□□□□□□□□□□□□□□any□all□list□□□□□□

□□□

□□□□□VS□□□□□□

□□□□□□

```
>>>G = (c * 4 for c in 'SPAM')
>>>I = iter(G)                                  # Iterate manually
>>>next(I)
'SSSS'
>>>next(I)
'PPPP'
>>>G = timesfour('spam')
>>>I = iter(G)
>>>next(I)
'ssss'
>>>next(I)
'pppp'
```

□□□

□□□□□□□□□□

□□□□□□□□□

```
>>>list(I1)                                  # Collect the rest of I1's items
['MMMM']
>>>next(I2)                                  # Other iterators exhausted too
StopIteration

>>>I3 = iter(G)                              # Ditto for new iterators
>>>next(I3)
StopIteration

>>>I3 = iter(c * 4 for c in 'SPAM')          # New generator to start over
>>>next(I3)
'SSSS'
```

□□□

```
>>>L = [1, 2, 3, 4]
>>>I1, I2 = iter(L), iter(L)
>>>next(I1)
1
>>>next(I1)
2
>>>next(I2)                          # Lists support multiple iterators
1
>>>del L[2:]                         # Changes reflected in iterators
>>>next(I1)
StopIteration
```

□□□

□□□□□□□zip□map

□□□□□□

□□map□zip□□□

```
# Using a list comprehension

def mymap(func, *seqs):
    return [func(*args) for args in zip(*seqs)]

print(mymap(abs, [-2, -1, 0, 1, 2]))
print(mymap(pow, [1, 2, 3], [2, 3, 4, 5]))
```

□□□

```
print(list(mymap(abs, [-2, -1, 0, 1, 2])))
print(list(mymap(pow, [1, 2, 3], [2, 3, 4, 5])))
```

□□□□□□□□□□□□□□□□list□□□□□□□□□□□□□□□□

```
# zip(seqs...) and 2.6 map(None, seqs...) workalikes
def myzip(*seqs):
    seqs = [list(S) for S in seqs]
    res = []
    while all(seqs):
        res.append(tuple(S.pop(0) for S in seqs))
    return res

def mymapPad(*seqs, pad=None):
    seqs = [list(S) for S in seqs]
    res = []
    while any(seqs):
        res.append(tuple((S.pop(0) if S else pad) for S in seqs))
    return res

S1, S2 = 'abc', 'xyz123'
print(myzip(S1, S2))
print(mymapPad(S1, S2))
print(mymapPad(S1, S2, pad=99))
```

□□□

```
[('a', 'x'), ('b', 'y'), ('c', 'z'), (None, '1'), (None, '2'), (None, '3')]
[('a', 'x'), ('b', 'y'), ('c', 'z'), (99, '1'), (99, '2'), (99, '3')]
```

□□□

```
# Alternate implementation with lengths

def myzip(*seqs):
    minlen = min(len(S) for S in seqs)
    return [tuple(S[i] for S in seqs) for i in range(minlen)]

def mymapPad(*seqs, pad=None):
    maxlen = max(len(S) for S in seqs)
    index = range(maxlen)
    return [tuple((S[i] if len(S) > i else pad) for S in seqs) for i in index]

S1, S2 = 'abc', 'xyz123'
print(myzip(S1, S2))
print(mymapPad(S1, S2))
print(mymapPad(S1, S2, pad=99))
```

□□□□□□□□len□□□□□□□□

□□□□□□□□□□□□list□□□□□□□□□□□□

□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□map□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□iters□□□□□□□□□□res□□□[]□□□□□□□□Python 3.0□□□□□□□□□□□□□□□list□□□□□□□□□□□□□□□□□□□□□□□□

```
def myzip(*args):
    iters = list(map(iter, args))
    ...rest as is...
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□map□□□□□□list□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
>>>for line in open('temp.txt'):
...     print(line, end='')
...
Tis but
a flesh wound.
```

□□□

Python 3.0□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□5□□□8□□□□□□□□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

```
>>>G = ((x, x * x) for x in range(10))
>>>next(G)
(0, 0)
>>>next(G)
(1, 1)
```

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

```
>>>{x + y for x in 'ab' for y in 'cd'}
{'bd', 'ac', 'ad', 'bc'}

>>>{x + y: (ord(x), ord(y)) for x in 'ab' for y in 'cd'}
{'bd': (98, 100), 'ac': (97, 99), 'ad': (97, 100), 'bc': (98, 99)}

>>>{k * 2 for k in ['spam', 'ham', 'sausage'] if k[0] == 's'}
{'sausagesausage', 'spamspam'}

>>>{k.upper(): k * 2 for k in ['spam', 'ham', 'sausage'] if k[0] == 's'}
{'SAUSAGE': 'sausagesausage', 'SPAM': 'spamspam'}
```

□□□

□□□□□□□□□□□□□□

□□□□□□□□□□□□□

```
# File mytimer.py

import time
reps = 1000
repslist = range(reps)

def timer(func, *pargs, **kargs):
    start = time.clock()
    for i in repslist:
        ret = func(*pargs, **kargs)
    elapsed = time.clock() - start
    return (elapsed, ret)
```

□□□

□□□

□□□□

□□□

□□□

□□□□

□□□□□□□□□□Python 3.0□□□□□□□□□□Windows Vista□□□□□□□□□□□□□——map□□□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
C:\misc>c:\python30\python timeseqs.py

3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)]
---------------------------------
forLoop : 2.64441 => [0...9999]
---------------------------------
listComp : 1.60110 => [0...9999]
---------------------------------
mapCall : 1.41977 => [0...9999]
---------------------------------
genExpr : 2.21758 => [0...9999]
---------------------------------
genFunc : 2.18696 => [0...9999]
```

□□□□□□□□□□□□□□□□□□□

□□□

```
def listComp():
    return [x + 10 for x in repslist]

def mapCall():
    return list(map((lambda x: x + 10), repslist))       # list in 3.0 only

def genExpr():
    return list(x + 10 for x in repslist)                # list in 2.6 + 3.0

def genFunc():
    def gen():
        for x in repslist:
        yield x + 10
    return list(gen())
...
...
```

□□□□□□map□□□□□□□□□□□□□□□□□□□□□□for□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□

```
C:\misc>c:\python30\python timeseqs.py
3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)]
---------------------------------
forLoop : 2.60754 => [10...10009]
---------------------------------
listComp : 1.57585 => [10...10009]
---------------------------------
mapCall : 3.10276 => [10...10009]
---------------------------------
genExpr : 1.96482 => [10...10009]
---------------------------------
genFunc : 1.95340 => [10...10009]
```

□□□□□□□□□□Python 2.5□□□□□□□□□□□□□□□□□□□

```
2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)]
forStatement          => 5.25699996948
listComprehension     => 2.68400001526
mapFunction           => 5.96900010109
generatorExpression   => 3.37400007248
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

```
# File mytimer.py (2.6 and 3.0)

"""
timer(spam, 1, 2, a=3, b=4, _reps=1000) calls and times spam(1, 2, a=3)
_reps times, and returns total time for all runs, with final result;

best(spam, 1, 2, a=3, b=4, _reps=50) runs best-of-N timer to filter out
any system load variation, and returns best time among _reps tests
"""

import time, sys
if sys.platform[:3] == 'win':
    timefunc = time.clock              # Use time.clock on Windows
else:
    timefunc = time.time               # Better resolution on some Unix platforms

def trace(*args): pass                 # Or: print args

def timer(func, *pargs, **kargs):
    _reps = kargs.pop('_reps', 1000)   # Passed-in or default reps
    trace(func, pargs, kargs, _reps)
    repslist = range(_reps)            # Hoist range out for 2.6 lists
    start = timefunc()
    for i in repslist:
        ret = func(*pargs, **kargs)
    elapsed = timefunc() - start
    return (elapsed, ret)

def best(func, *pargs, **kargs):


 _reps = kargs.pop('_reps', 50)
 best = 2 ** 32
 for i in range(_reps):
     (time, ret) = timer(func, *pargs, _reps=1, **kargs)
     if time < best: best = time
 return (best, ret)
```

□□□

```
# File timeseqs.py

import sys, mytimer
reps = 10000
repslist = range(reps)

def forLoop(): ...

def listComp(): ...

def mapCall(): ...

def genExpr(): ...

def genFunc(): ...

print(sys.version)
for tester in (mytimer.timer, mytimer.best):
    print('<%s>' % tester.__name__)
    for test in (forLoop, listComp, mapCall, genExpr, genFunc):
        elapsed, result = tester(test)
        print ('-' * 35)
        print ('%-9s: %.5f => [%s...%s]' %
               (test.__name__, elapsed, result[0], result[-1]))
```

□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□N□□□□N□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

```
-----------------------------------
genExpr : 1.97440  => [10...10009]
-----------------------------------
genFunc : 1.95072  => [10...10009]
<best>
-----------------------------------
forLoop : 0.00193  => [10...10009]
-----------------------------------
listComp : 0.00124 => [10...10009]
-----------------------------------
mapCall : 0.00268  => [10...10009]
-----------------------------------
genExpr : 0.00164  => [10...10009]
-----------------------------------
genFunc : 0.00165  => [10...10009]
```

□□□□□N□□□□□

```
# File mytimer.py (3.X only)

"""
Use 3.0 keyword-only default arguments, instead of ** and dict pops.
No need to hoist range() out of test in 3.0: a generator, not a list
"""

import time, sys
trace = lambda *args: None # or print
timefunc = time.clock if sys.platform == 'win32' else time.time

def timer(func, *pargs, _reps=1000, **kargs):
    trace(func, pargs, kargs, _reps)
    start = timefunc()
    for i in range(_reps):
        ret = func(*pargs, **kargs)
    elapsed = timefunc() - start
    return (elapsed, ret)

def best(func, *pargs, _reps=50, **kargs):
    best = 2 ** 32
    for i in range(_reps):


        (time, ret) = timer(func, *pargs, _reps=1, **kargs)
        if time < best: best = time
   return (best, ret)
```

□□□

```
C:\misc>c:\python30\python
>>>from mytimer import timer, best
>>>
>>>def power(X, Y): return X ** Y                # Test function
...
>>>timer(power, 2, 32)                           # Total time, last result
(0.002625403507987747, 4294967296)
>>>timer(power, 2, 32, _reps=1000000)            # Override defult reps
 (1.1822605247314932, 4294967296)
>>>timer(power, 2, 100000)[0]                    # 2 ** 100,000 tot time @1,000 reps
2.2496919999608878

>>>best(power, 2, 32)                            # Best time, last result
(5.58730229727189e-06, 4294967296)
>>>best(power, 2, 100000)[0]                     # 2 ** 100,000 best time
0.0019937589833460834
>>>best(power, 2, 100000, _reps=500)[0]          # Override default reps
0.0019845399345541637
```

□□□

□□□19□□□□□Python 3.0□□keyword-only□□□□□

□□□□

□□Python□□□□□□□□□□ timeit□□□Python□□□□□□□□□□

□□□□□□□□□profile□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□35□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□

□□□

```
>>>def selector():
...     print(X)                    # Does not yet exist!
...     X = 88                      # X classified as a local name (everywhere)
...                                 # Can also happen for "import X", "def X"...
>>>selector()
...error text omitted...
UnboundLocalError: local variable 'X' referenced before assignment
```

□□□

□□□

□□□□□□□

□□□□□□□def□□□□□□□

□□□

□□□

□□return□□□□□

□Python□□□□return□□□yield□□□□□

□□□□15□□□□□“□□□□□□□□□”□□□□□□□□□

□□□□□□□□□□

□□□□17□□□□□□□□□

□□□□

□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□

3.□□□□□□□□□□□□□□□□□□□

4.yield□□□□□□□□□

5.map□□□list comprehension□□□□□□□□□□□□□□□□□□

□□□□

1.□□□

□□□

□□□□ □□□

□□□

□□□□□□□□□□□□□□3□□□□□□□□□□□□□□□□□□□□□□□□□□□□□dict.keys□□□□□□□□□□□□□□□for□while□□□□□□□□□Python 3.0□□□□□□□□□list□□□□□

5.□□□□□□copyDict(dict)□□keys□□□□□□□□□□□□□□Python 2.2□□□□□□□□□□□□□□□□□□□□keys□□□□□□□□□□□□X[:]□□□□□□□□□□□□□□□□□□□□□□□□

6.□□□□□□addDict(dict1,dict2)□□□□□□□□□□□□□□□□□□□

```
>>>f1(1, 2)
>>>f1(b=2, a=1)

>>>f2(1, 2, 3)
>>>f3(1, x=2, y=3)
>>>f4(1, 2, 3, x=2, y=3)

>>>f5(1)
>>>f5(1, 4)

>>>f6(1)
>>>f6(1, 3, 4)
```

8.□□□□□□□□13□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
x = y // 2                          # For some y > 1
while x > 1:
    if y % x == 0:                  # Remainder
      print(y, 'has factor', x)
      break                         # Skip else
    x -= 1
else:                               # Normal exit
    print(y, 'is prime')
```

□□□□□□□□□□□□□□□□□□□□□□□□□□y□□□/□□□□□□//□□□□□□□□□□□□□□□□□Python 3.0□/□□□□□□□□□□□□□□□□□□□□□5□□□□□□□□□□□□□□□□□□□□0□1□□□□□□□□□□□□□□□□□□□

```
13 is prime
13.0 is prime
15 has factor 5
15.0 has factor 5.0
```

9.□□□

□□□□ □□

□21□ □□□□□□□□

□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□

□□reload□__name__□__all__□□□□□□import□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□□”□□□□□□□□□□□□□□□□□□□

Python□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□

□□□□□

□□□□□□□□□□□□21-1□□□□□□□□□□□□Python□□□□□□a.py□b.py□c.py□□□a□□□□□□□□□□□□

```
def spam(text):
    print(text, 'spam')
```

□□□□□a.py□□□□spam□□□□□□□□□□a.py□□□□□□□□□□□□□□□□□Python□□□

```
import b
b.spam('gumby')
```

□□□□□Python import□□□□□□□a.py□□□□□□b.py□□□□□□□□□□□□□□□□□

□Python□□□□□□□□object.attribute□□□□□□□□

□□□□□

□□□21-1□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□200□□□□□□□□□□□□□□□□□□□

import□□□□

□□□□

□□□Python□□□□□import□□□□□

2.□□□□□□

□□□□□□□□□□□□□□import□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□2□□□□□□□□□□

Python□□□□□□□□□□□□□□□□□□□

3.□□

import□□□

□□□□□□

□□□□□

Python□□□

.pth□□□□

□□□Python□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□PYTHONPATH□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□

sys.path□□

□□□

[1]□□□□□□□□□□□□□□□□sys.path□□□□□□□□□□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□.py□□□□□import□□□□□□□Python□□□□□□□□□□□□□□□

C□□□Jython□□□

□□□□□□□□□□

□□□□□□□□□

□□□□distutils□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□

□□□□

□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□PYTHONPATH□□□□□□

3.□□□□□□□□□□□□□□□□□□□□□□

4.□□Python□□□□□□□□□□□import□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□

□22□ □□□□□□□□

□□□□□□□

□□□□□□Python□□□□□□□□□□□□.py□□□□□□□□□□11□□□□□□□□□□□□

□□□□□

□□□□□□□□import□from□□□□□□□□

from□□

□□from□□□

from*□□

□□□□□□□□□□□□□□from□□□□□□□□*□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□□□□□□□□□□?”□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□spam□□□□□□□□□□

```
>>>simple.spam = 2      # Change attribute in module
>>>import simple        # Just fetches already loaded module
>>>simple.spam          # Code wasn't rerun: attribute unchanged
2
```

□□□□□□□□□□□□□□□□□□□

import□from□□□□□

□□def□□□import□from□□□□□

```
>>>import small                    # Get module name (from doesn't)
>>>small.x                         # Small's x is not my x
1
>>>small.y                         # But we share a changed mutable
[42, 2]
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□18-1□□□□□□□□□□□□□□□□□“□□□”□“□□”□□“□□□□□”□“□□□”□□Python□□□□□□□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□x□□□□□□□□□□□□□□□□□□□x□□□□□□□□□□□x□□from□□□□□□□□□□□□□□□□□□□

import□from□□□□

□□□□□□□□□□□□□□□□□□from□□□import□□□□□□□small□□□□□□□□from□□□□□□□□□□□□□□□□□□

from□□□□□□□

□□from□□□□□□□□□□□□□□□□□□□□module.name□□□name□□□□□□□□□□□□□□□Python□□□□□□□□□□□import□□□from□□□□□□□□□□□□□□□□□□□□from□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□from*□□□□□□□□□□□from□□□□□□□□□□□

```
# O.py

import M, N                    # Get the whole modules, not their names
M.func()                       # We can call both names now
N.func()                       # The module names make them unique
```

□□□

□□□□□□

□□□□□

·□□□□□□□□□□□□□□□__dict__□dir(M)□□□□□□□□□□□□□□□

print□□□□□□□□□□□

```
>>>import module2
starting to load...
done loading.
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□import□□□□□□□□□□□□□□□□□□□

□□——□□□□□□□□□□

```
>>>list(module2.__dict__.keys())
['name', '__builtins__', '__file__', '__package__', 'sys', 'klass', 'func',
'__name__', '__doc__']
```

□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

X.Y.Z□□□□□□□□X□□□□□□□Y□□□□□□□□X.Y□□□Z□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□

```
% python modb.py
11 99
```

□□□

□□□□□□□

□□□□□□

□□□□□□□□□mod1□□mod2□□□□□□□□□

□□□□

□□□

□□□□□□□□□□□□Python 2.6□□reload□□□□□□□□□□□□Python 3.0□□□□□□□□imp□□□□□□——□Python 3.0□□□imp.reload□□□□□□□□□□□□□□import□□□from□□□□□□□□□□□Python 3.0□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——Python 2.6□□imp□□□□□□□reload□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□

reload□□

□import□from□□□□□

·reload□Python□□□□□□□□□□□□□

·□□reload□□□□□□□□□□□□□□□□□□□□□□

·reload□Python 3.0□□□□□□□□□□□□□□□□□□

□□reload□□□□□□□□□□□□□□□□□□□

```
import module                   # Initial import
...use module.attributes...
...                             # Now, go change the module file
...
from imp import reload          # Get reload itself (in 3.0)
reload(module)                  # Get updated exports
...use module.attributes...
```

□□□

reload□□

□□□□□□□□□reload□□□□□□□□□

```
message = "After editing"
def printer():

print('reloaded:', message)
```

□□□□□Python□□□□□

□□□GUI□□□□□□□□□□□□□□□□□□□□□□GUI□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□import□from□□□□□reload□□□□□□□□from□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□

2.from□□□import□□□□□□□□

3.reload□□□□□□□□□□□□

4.□□□□□□□□□import□□□□□from□

5.□□□□from□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□

□23□ □□□

□□□Python□□□□□□□□□□□□□□□□□□□

from□□□□□□□□□

```
from dir1.dir2.mod import x
```

□□□□□□“□□”□□□□□□□□□

□□□□□□□□□□dir0□□□□□□□□□□

__init__.py□□□

□□□

```
dir0\                    # Container on module search path
    dir1\
        __init__.py
        dir2\
            __init__.py
            mod.py
```

__init__.py□□□□Python□□□□

□□□

□□□□□

□□□□

```
% python
>>>import dir1.dir2.mod                    # First imports run init files
dir1 init
dir2 init
in mod.py
>>>
>>>import dir1.dir2.mod                    # Later imports do not
>>>
>>>from imp import reload                  # Needed in 3.0
>>>reload(dir1)
dir1 init
<module 'dir1' from 'dir1\__init__.pyc'>
>>>
>>>reload(dir1.dir2)
dir2 init
<module 'dir1.dir2' from 'dir1\dir2\__init__.pyc'>
```

□□□□import□□□□□□□□□□□□□□□□□□□□□□□□□□□□mod□□□□□□□□□□ dir2□□□dir2□□□□□□dir1□□

```
>>>dir1
<module 'dir1' from 'dir1\__init__.pyc'>
>>>dir1.dir2
<module 'dir1.dir2' from 'dir1\dir2\__init__.pyc'>
>>>dir1.dir2.mod
<module 'dir1.dir2.mod' from 'dir1\dir2\mod.pyc'>
```

□□□□□□□□□□□□□□□□□□□

```
>>>dir1.x
1
>>>dir1.dir2.y
2
>>>dir1.dir2.mod.z
3
```

□□□□from□□□import□□

import□□□

```
% python
>>>from dir1.dir2 import mod              # Code path here only
dir1 init
dir2 init
in mod.py
>>>mod.z                                  # Don't repeat path
3
>>>from dir1.dir2.mod import z
>>>z
3
>>>import dir1.dir2.mod as mod            # Use shorter name (see Chapter 24)
>>>mod.z
3
```

□□□□□□□□□

□□□□Python□□□□□□□□

□□□

□□system1\main.py□□□□□□□□□□□□□□□□□□□

```
root\
    system1\
        __init__.py
        utilities.py
        main.py
        other.py
    system2\
        __init__.py
        utilities.py
        main.py
        other.py
    system3\                        # Here or elsewhere
        __init__.py                 # Your new code here
        myfile.py
```

□□□

□□□

□□□□□

□□□□

□□□□□Python 3.0□□□□□□□□□□□□from□□□□□□□Python 3.0□□□□□□□□□□□□

□□□□□□□□

□Python 3.0□Python 2.6□□from□□□□□□□□□□□□□□(“.”)□□□□□□□□□□□□□□

□□□“□□□spam□□□□□□□□name□□□□spam□□□□□□□□□□□□□□□□□□□□□□□□□□□□”

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□Python 2.6□□□□□□□import□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□from□□□□□□□□□□□□□□import□□□□□□□Python 3.0□□import modname□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

□□□□Python□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
from .string import name1, name2          # Imports names from mypkg.string
```

□□□□□□□□□□□□□□□□□string□□□□□□□□□□□□□□□□□mypkg.main□□□□□□□□mypkg.string□□name1□name2□

□□□□□□□□□□□“.”□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□

□□□□□□□□

□□□□sys.path□□□□

□□□□□□□□□

□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□

·□□□□□□□□□□A□□□□□sys.path□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□“□”□□□□□□□□□□□□□□□□

```
C:\test>del string*
C:\test>mkdir pkg

# test\pkg\spam.py
import eggs                          # <== Works in 2.6 but not 3.0!
print(eggs.X)

# test\pkg\eggs.py
X = 99999
import string
print(string)
```

□□□□□□□□□□□□□□□□□□□□□□import□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□Python 3.0□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□

```
C:\test>c:\Python26\python
>>>import pkg.spam
<module 'string' from 'c:\Python26\lib\string.pyc'>
99999

C:\test>c:\Python30\python
>>>import pkg.spam
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pkg\spam.py", line 1, in <module>
    import eggs
ImportError: No module named eggs
```

□□□□□□Python 2.6□Python 3.0□□Python 3.0□□□□□□□□□

```
# test\pkg\spam.py
from . import eggs                          # <== Use package relative import in 2.6 or 3.0
print(eggs.X)

# test\pkg\eggs.py
X = 99999
import string
print(string)

C:\test>c:\Python26\python
>>>import pkg.spam
<module 'string' from 'c:\Python26\lib\string.pyc'>
99999

C:\test>c:\Python30\python
>>>import pkg.spam
<module 'string' from 'c:\Python30\lib\string.py'>
99999
```

□□□□□□□□CWD□

□□□□□□□□□□□□□□□□□□□□□□string□□□□□□□□□□□□□□□□□□□

```
# test\string.py
print('string' * 8)

# test\pkg\spam.py
from . import eggs
print(eggs.X)

# test\pkg\eggs.py
X = 99999
import string                          # <== Gets string in CWD, not Python lib!
print(string)

C:\test>c:\Python30\python             # Same result in 2.6
>>>import pkg.spam
stringstringstringstringstringstringstringstring
<module 'string' from 'string.py'>
99999
```

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

```
C:\test>c:\Python30\python
>>>import pkg.spam
<module 'string' from 'c:\Python30\lib\string.py'>

C:\test>c:\Python26\python
>>>import pkg.spam
NiNiNiNiNiNiNiNi
<module 'pkg.string' from 'pkg\string.py'>
```

□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□——□□□Python 3.0□□□□□□□□□□□□□□□□□□□

```
# test\pkg\spam.py
from . import string                    # <== Fails if no string.py here!

C:\test>C:\python30\python
>>>import pkg.spam
...text omitted...
ImportError: cannot import name string
```

□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□CWD□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□sys.path□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□

□□□

```
from email.message import Message
from tkinter.filedialog import askopenfilename
from http.server import CGIHTTPRequestHandler

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

□□□□

□□□□□Python□□□□□

□□□□

1.□□□□□□□□__init__.py□□□□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3.□□□□□□□__init__.py□□□

4.□□□□□□□□□□□□import□□□□□□from□□□□

5.from mypkg import spam□from.import spam□□□□□□

□□□□

1.__init__.py□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

第24章　□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__future__□□□__name__□□□sys.path□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□

□□□from*□□□□□_X□__all__

□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□□□__name__□__main__

□□□□□□

```
% python runme.py
It's Christmas in Heaven...
```

□□□□□□□□□□__name__□□□□□□□□□□

```
def minmax(test, *args):
    res = args[0]
    for arg in args[1:]:
        if test(arg, res):
            res = arg
    return res

def lessthan(x, y): return x < y
def grtrthan(x, y): return x > y

print(minmax(lessthan, 4, 2, 1, 5, 6, 3))               # Self-test code
print(minmax(grtrthan, 4, 2, 1, 5, 6, 3))
```

□□□

□□□□□□□□□□□□□□□□□□

```
% python min.py
I am: __main__
1
6
```

□□□□□□□□□□□□□□□□□□□□□□□__main__□□□□□□□□□□□□□□□□□□

```
>>>import min
I am: min
>>>min.minmax(min.lessthan, 's', 'p', 'a', 'm')
'a'
```

□□□□□□□□□□□□□□□□□□□

□□□□__name__□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□__name__□□□□□□□□□□□□□□□□ formats.py□□□□□□□□□□□□□□□□□□□

```
    fract = ('%.2f' % N)[-2:]
    format = '%s%s.%s' % (sign, whole, fract)
    return '$%*s' % (width, format)

if __name__ == '__main__':
    def selftest():
        tests = 0, 1          # fails: -1, 1.23
        tests += 12, 123, 1234, 12345, 123456, 1234567
        tests += 2 ** 32, 2 ** 100
        for test in tests:
            print(commas(test))

        print('')
        tests = 0, 1, -1, 1.23, 1., 1.2, 3.14159
        tests += 12.34, 12.344, 12.345, 12.346
        tests += 2 ** 32, (2 ** 32 + .2345)
        tests += 1.2345, 1.2, 0.2345
        tests += -1.2345, -1.2, -0.2345
        tests += -(2 ** 32), -(2**32 + .2345)
        tests += (2 ** 100), -(2 ** 100)
        for test in tests:
            print('%s [%s]' % (money(test, 17), test))

    import sys
    if len(sys.argv) == 1:
        selftest()
    else:
        print(money(float(sys.argv[1]), int(sys.argv[2])))
```

□□□□□Python 2.6□Python 3.0□□□□□

```
C:\misc>python formats.py 999999999 0
$999,999,999.00

C:\misc> python formats.py -999999999 0
$-999,999,999.00

C:\misc>python formats.py 123456789012345 0
$123,456,789,012,345.00

C:\misc> python formats.py -123456789012345 25
$ -123,456,789,012,345.00

C:\misc>python formats.py 123.456 0
$123.46

C:\misc> python formats.py -123.454 0
$-123.45

C:\misc>python formats.py
...canned tests: try this yourself...
```

□□□

```
>>>from formats import money, commas
>>>money(123.456)
'$123.46'
>>>money(-9999999.99, 15)
'$ -9,999,999.99'
>>>X = 99999999999999999999
>>>'%s (%s)' % (commas(X), X)
'99,999,999,999,999,999,999 (99999999999999999999)'
```

□□□□□□□□□□15□□□□□□□□□□□□□□□□□□□□□□□□help□□□□□□□□□——□□□□□□□□□□□□□

```
>>>import formats
>>>help(formats)
Help on module formats:

NAME
    formats

FILE
    c:\misc\formats.py

DESCRIPTION
    Various specialized string display formatting utilities.
    Test me with canned self-test or command-line arguments.

FUNCTIONS
    commas(N)
        format positive integer-like N for display with
        commas between digit groupings: xxx,yyy,zzz

    money(N, width=0)
        format number N for display with commas, 2 decimal digits,
        leading $ and sign, and optional padding: $ -xxx,yyy.zz
```

□□□

□□□□□□□□

□□21□□□□□□□

□□□□□sys.path□□□□□□□□□□Python□□□□□□□□□□

Import□□□from□□□as□□

import□from□□□

□□□□□□23□□□□□□

□□□□□□□□□□

□□□□□□□□□

```
"""
mydir.py: a module that lists the namespaces of other modules
"""

seplen = 60
sepchr = '-'

def listing(module, verbose=True):
    sepline = sepchr * seplen
    if verbose:
        print(sepline)
        print('name:', module.__name__, 'file:', module.__file__)
        print(sepline)

    count = 0
    for attr in module.__dict__:                # Scan namespace keys
        print('%02d) %s' % (count, attr), end = ' ')
        if attr.startswith('__'):
            print('<built-in name>')            # Skip __file__, etc.
        else:
            print(getattr(module, attr))        # Same as .__dict__[attr]
        count += 1

    if verbose:
        print(sepline)
        print(module.__name__, 'has %d names' % count)
        print(sepline)

if __name__ == '__main__':
    import mydir
    listing(mydir)                              # Self-test code: list myself
```

□□□□□□□□□□□□□□□□□□□formats.py□□□□□□□□□□□□□□□□□□□

```
NAME
     mydir - mydir.py: a module that lists the namespaces of other modules

FILE
    c:\users\veramark\mark\mydir.py

FUNCTIONS
    listing(module, verbose=True)

DATA
    sepchr = '-'
    seplen = 60
```

□□□

```
>>>import mydir
>>>import tkinter
>>>mydir.listing(tkinter)
------------------------------------------------------------
name: tkinter file: c:\PYTHON30\lib\tkinter\__init__.py
------------------------------------------------------------
00) getdouble <class 'float'>
01) MULTIPLE multiple
02) mainloop <function mainloop at 0x02913B70>
03) Canvas <class 'tkinter.Canvas'>
04) AtSelLast <function AtSelLast at 0x028FA7C8>
...many more name omitted...
151) StringVar <class 'tkinter.StringVar'>
152) ARC arc
153) At <function At at 0x028FA738>
154) NSEW nsew
155) SCROLL scroll


------------------------------------------------------------
tkinter has 156 names
------------------------------------------------------------
```

□□□□□□□□getattr□□□□□□□□□□□□□□□□□□□□mydir□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□[2]□

[1]□□17□□□□□□□□□□□□□□□□□□□□□□sys.modules□□□□□□□□□□□□□□□□□□□

[2] mydir.listing PYTHONSTARTIP __main__ A

□□□□□□□□□□

□□import□from□□□□□□□□□

□exec□Python 2.6□□□

□□□□□□□

□□□□22□□□□□□

□□□□□□□□□□reloadall.py□□□reload_all□□□□□□□□□□

□□Python 3.0□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□print□□□□+□□□□□□□□□□

```
C:\misc>c:\Python30\python reloadall.py
reloading reloadall
reloading types
```

□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□o s□□□□tkinter□□□□□tkinter□os□□□□□□□□sys□□□□□□□Python 2.6□□□□□□□□□□Tkinter□□tkinter□□

```
>>>from reloadall import reload_all
>>>import os, tkinter

>>>reload_all(os)
reloading os
reloading copyreg
reloading ntpath
reloading genericpath
reloading stat
reloading sys
reloading errno

>>>reload_all(tkinter)
reloading tkinter
reloading _tkinter
reloading tkinter._fix
reloading sys
reloading ctypes
reloading os
reloading copyreg
reloading ntpath
reloading genericpath
reloading stat
reloading errno
reloading ctypes._endian
reloading tkinter.constants
```

□□□

```
import b                                    # a.py
X = 1

import c                                    # b.py
Y = 2

Z = 3                                       # c.py
```

```
C:\misc>C:\Python30\python
>>>import a
>>>a.X, a.b.Y, a.b.c.Z
(1, 2, 3)
```

Change all three files' assignment values and save

```
>>>from imp import reload
>>>reload(a)                               # Normal reload is top level only
<module 'a' from 'a.py'>
>>>a.X, a.b.Y, a.b.c.Z
(111, 2, 3)

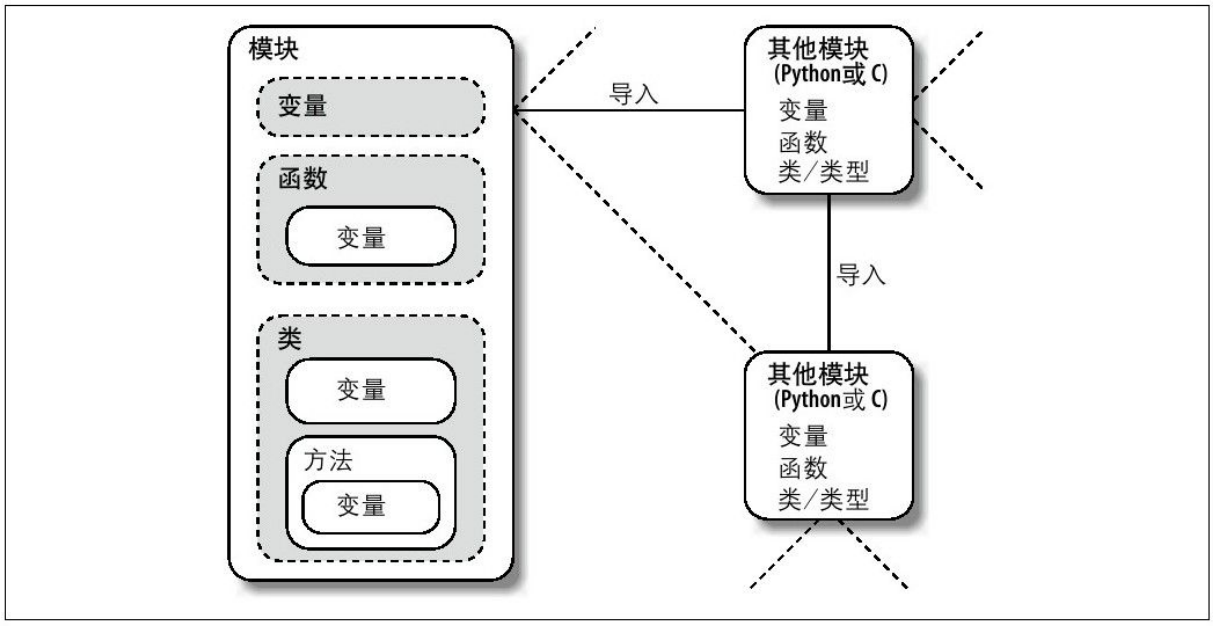
>>>from reloadall import reload_all
>>>reload_all(a)
reloading a
reloading b
reloading c


>>>a.X, a.b.Y, a.b.c.Z                    # Reloads all nested modules too
(111, 222, 333)
```

□□□

□□□□□□

□□□□□


模块
变量
函数
变量
类
变量
方法
变量
导入
其他模块
(Python或C)
变量
函数
类/类型
导入
其他模块
(Python或C)
变量
函数
类/类型

□ 24-1 □□□□

□□□□

□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□

```
func1()                          # Error: "func1" not yet assigned

def func1():
    print(func2())               # Okay: "func2" looked up later

func1()                          # Error: "func2" not yet assigned

def func2():
    return "Hello"

func1()                          # Okay: "func1" and "func2" assigned
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□func1□□□□□□□□□□func1 def□□□□□□□□func1□□□□func2□def□□□□□□□func1□□func2□□□□□□□□□□□□□□□□□□□func1□□□□□□□fun2□def□□□□□□□□□□□□□□□□□□func1□□□□□□□□□□□□func1□func2□□□□□□□□

□□□□□□□□□def□□□□□□□□□□□□□□□□□□□

from□□□□□□□□□□□□

□□□□□□□from□□□□Python□□□□□□□□□□□□from□□□□□□□□□□□□

```
# nested3.py
import nested1                          # Get module as a whole
nested1.X = 88                          # OK: change nested1's X
nested1.printer()

% python nested3.py
88
```

from*□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□from module import*□□□□□□□□□□□□□□□□□□□

reload□□□□from□□

□□□□□□from□□□□□□□□□□□□□□□□□□□□from□□□from□□□□□□□□□□□□□□□□□□□

reload□from□□□□□□□□

□3□□□□□□□□

function□□□□reload□□□□module.function□□□□□□□from□

```
from imp import reload
import module
reload(module)
from module import function                    # Or give up and use module.function()
function(1, 2, 3)
```

□□□□□□□function□□□□□□□□

□□□□□□□□□□reload□from□□□□□□□□□□

□□□□□from□□□□□□

□□□□□□

```
# recur1.py
X = 1
import recur2                          # Run recur2 now if it doesn't exist
Y = 2

# recur2.py
from recur1 import X                   # OK: "X" already assigned
from recur1 import Y                   # Error: "Y" not yet assigned
```

```
C:\misc>C:\Python30\python
>>>import recur1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "recur1.py", line 2, in <module>
    import recur2
  File "recur2.py", line 2, in <module>
    from recur1 import Y
ImportError: cannot import name Y
```

□recur1□□□□recur2□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□recur2□□□□recur1□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□from□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

·□□□

·□□□□□□□□□□□□□□□□□□□□import□□□□□□□□□□from□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__future__□□□□□□□□□□□□□__name__□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□__name__□□□□□□"__main__"□□□□□□□□□□□

3.□□

4.□□sys.path□□□PYTHONPATH□□□□□□□□□□□□□□□□□□

5.□□□□__future__□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□from*□□□□□□□□□□□□□□□□□□□

□□□□ □□□

□□□□B□“□□□□□□”□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□UNIX□wc□□□□□□□□□□□□□□□□□□□□mymod.py□Python□□□□□□□□□□□□□□□

·countLines(name)□□□□□□□□□□□□□□□□□□□□□□□□□□□□file.readlines□□□□□□□□□□□□□□□□□□□□□□□□□□□□len□□□□

·countChars(name)□□□□□□□□□□□□□□□□□□□□□□□□□□□□file.read□□□□□□□□□□□□□

·test(name)□□□□□□□□□□□□□□□□□□□

test("mymod.py")□□□

6.□□□□□□□□□□□□□□□□22□changer.py□□□□□□□□□□□□□□□□

□□□□ □□OOP

□25□ OOP□□□□□

□□□□□□□□□□□□“□□”□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□OOP

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□OOP□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

□Python□□□□□□□□□□□□□□□□□□□

□□

□□□

□□□

□□□□

□□□Python□□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□I2.w□□□□□□Python□□□□□□□□□□C3.w□□□□□□□□□□□□□□□□□□□

□□□□

□□class□□□□□□□□□□□□□□□□□□□

□□□

□□□

□□□

OOP□□□□□□□

□□□Python□OOP□□□□□□□

```
class Employee:                          # General superclass
    def computeSalary(self): ...         # Common or default behavior
    def giveRaise(self): ...
    def promote(self): ...
    def retire(self): ...
```

□□□

```
company = [bob, mel]                    # A composite object
for emp in company:
    print(emp.computeSalary())          # Run this object's version
```

□□□4□□□16□□□

```
class Reader:
    def read(self): ...                 # Default behavior and tools
    def other(self): ...
class FileReader(Reader):
    def read(self): ...                 # Read from a local file
class SocketReader(Reader):
    def read(self): ...                 # Read from a network socket
...
processor(FileReader(...), Converter, FileWriter(...))
processor(SocketReader(...), Converter, TapeWriter(...))
processor(FtpReader(...), Converter, XmlWriter(...))
```

□□□

□□□

□□□□

□□□□□OOP□□□□□

□□□□

1.Python□OOP□□□□□□□□□□

2.□□□□□□□□□□□□

3.□□□□□□□□□□□□□□

4.□□□□□□□□□□□□□□□□□□□□

5.__init__□□□□□□□□□

6.□□□□□□□□

7.□□□□□□

8.□□□□□□□□□

□□□□

1.OOP□□□

7.□□□□□class□□□□□

第26章　□□□□□□□

□□□□□□□□□□□□□□OOP□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□

□□□□□□□□□OOP□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□Python OOP□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□def□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□

□□□□

```
>>> x = FirstClass()                    # Make two instances
>>> y = FirstClass()                    # Each is a new namespace
```

□□□

□FirstClass□setdata□□□□□□□□□□□□self.data□□□□□□self□□□□□□□□□□□□

□□□

□□□□□□□□□

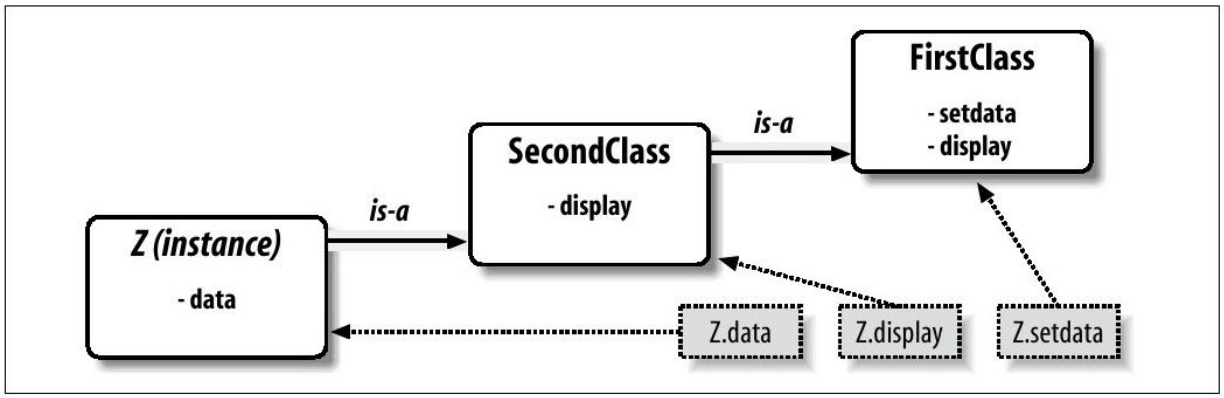
□□□□□□□□

□□□

□□

```
>>> z = SecondClass()
>>> z.setdata(42)                    # Finds setdata in FirstClass
>>> z.display()                      # Finds overridden method in SecondClass
Current value = "42"
```

□□□□□□□□□□□Second Class□□□□□□□□□set data□□□□□ FirstClass□□□□□□□□□□display□□□□□SecondClass□□□□□□□□□□□ 26-2□□□□□□□□□□□□□□



□ 26-2 □□□□□□□□□□□□□□□

□□□□□□□FirstClass□□□□□□□

□□□□□□□□

□□□□□□□

```
# food.py
var = 1                                  # food.var
def func():                              # food.func
    ...
class spam:                              # food.spam
    ...
class ham:                               # food.ham
    ...
class eggs:                              # food.eggs
    ...
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□person.py□□□□□□□

```
class person:
    ...
```

□□□□□□□□□□□□□□□□□

```
import person                            # Import module

x = person.person()                      # Class within module
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□person.person□□□person□□□□person□□□□person□□□□person□□□□□□□□□□□□□□□□□□□□□□from□□□

```
from person import person                # Get class from module
x = person()                             # Use class name
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

```
import person                                    # Lowercase for modules
x=person.person()                                # Uppercase for classes
```

□□□□□□□□□□□□□□□□□□□

□□□□□Python□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>> class ThirdClass(SecondClass):                    # Inherit from SecondClass
...     def __init__(self, value):                    # On "ThirdClass(value)"
...         self.data = value
...     def __add__(self, other):                     # On "self + other"
...         return ThirdClass(self.data + other)


...     def __str__(self):                            # On "print(self)", "str()"
...         return '[ThirdClass: %s]' % self.data
...     def mul(self, other):                         # In-place change: named
...         self.data *= other
...
>>> a = ThirdClass('abc')       # __init__ called
>>> a.display()                 # Inherited method called
Current value = "abc"
>>> print(a)                    # __str__: returns display string
[ThirdClass: abc]

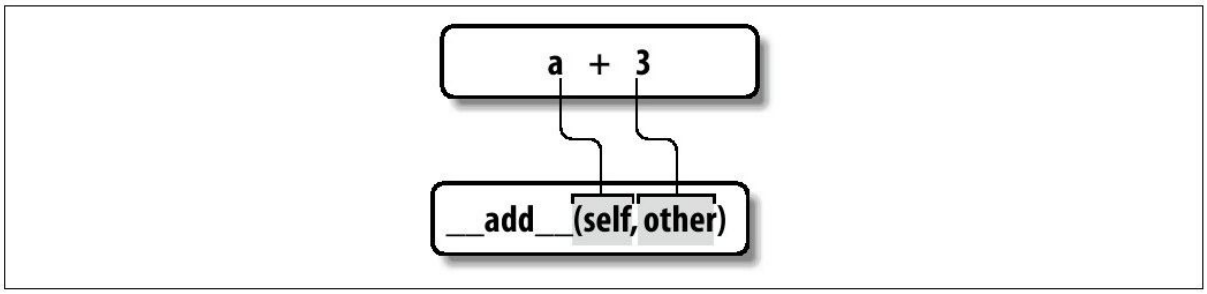
>>> b = a + 'xyz'               # __add__: makes a new instance
>>> b.display()                 # b has all ThirdClass methods
Current value = "abcxyz"
>>> print(b)                    # __str__: returns display string
[ThirdClass: abcxyz]

>>> a.mul(3)                    # mul: changes instance in-place
>>> print(a)
[ThirdClass: abcabcabc]
```

ThirdClass“□□□”SecondClass□□□□□□□□□□□□SecondClass□display□□□□□□Third Class□□□□□□□□□□□□□□□□□□□□□"abc"□□□□□□__init__□□□□□□□□value□□□□□□□□self.data□□□□□□□ThirdClass□□□□□□□□□data□□□□□□□□□□□□□□□setdata□□□

□□□ThirdClass□□□□□□□□□□+□□□□print□□□□□□+□Python□□□□□□□□□□__add__□□self□□□□□□□□□□□□other□□□26-3□□□__add__□□□□□□□+□□□□□□□□□□print□Python□□□□□□□□□□__str__□□self□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□__str__□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□□□□display□□□



□ 26-3 □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□Python□

□□□□□□□□□□class□□□□□□□□□□

```
>>> x = rec()                           # Instances inherit class names
>>> y = rec()
```

□□□

```
>>> rec.__dict__.keys()
['__module__', 'name', 'age', '__dict__', '__weakref__', '__doc__']

>>> list(x.__dict__.keys())
['name']

>>> list(y.__dict__.keys())                # list() not required in Python 2.6
[]
```

□□□□□□□□□□□□□□□□□□□□□□name□age□□□x□□□□name□□y□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□——□□□□□□□□□□□□□□

□□□

```
>>> class rec: pass
...
>>> pers1 = rec()                                  # Instance-based records
>>> pers1.name = 'mel'
>>> pers1.job = 'trainer'
>>> pers1.age = 40
>>>
>>> pers2 = rec()
>>> pers2.name = 'vls'
>>> pers2.job = 'developer'
>>>
>>> pers1.name, pers2.name
('mel', 'vls')
```

□□□

□□□

□□□□

□□□□Python□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□

2.□□□□□□□□□□□□

3.□□□□□□□□□□□□□□□□□□□□□□

4.□□□□□□□□□□□□□□□□□□□□□□□□

5.Python□□□self□□□□□□

6.Python□□□□□□□□□□□□

7.□□□□□□□□□□□□□□□□□□□

8.□□□□□□□□□□□□□□□□

9.Python OOP□□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□

6.Python□□□

□27□　□□□□

□□□□

□□□□

□□1□□□□□

□□□Person□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__init__□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
# Add record field initialization

class Person:
    def __init__(self, name, job, pay):                # Constructor takes 3 arguments
        self.name = name                               # Fill out fields when created
        self.job = job                                 # self is the new instance object
        self.pay = pay
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□pay□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□

□□□□□□bob□□□□job□pay□□□□□□□□□□sue□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

```
C:\misc> person.py
Bob Smith 0
Sue Jones 100000

c:\misc> python
Python 3.0.1 (r301:69561, Feb 13 2009, 20:04:18) ...
>>> import person
>>>
```

□□□

```
print('{0} {1}'.format(bob.name, bob.pay))          # New format method
print('%s %s' % (bob.name, bob.pay))                # Format expression
```

□□2□□□□□□□

□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□Person□□□□□□□□□□□□□□name□pay□□□□□□□□□□
bob.name□sue.pay□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
# Process embedded built-in types: strings, mutability

class Person:
    def __init__(self, name, job=None, pay=0):
        self.name = name
        self.job = job
        self.pay = pay

if __name__ == '__main__':
    bob = Person('Bob Smith')
    sue = Person('Sue Jones', job='dev', pay=100000)
    print(bob.name, bob.pay)
    print(sue.name, sue.pay)
    print(bob.name.split()[-1])          # Extract object's last name
    sue.pay *= 1.10                      # Give this object a raise
    print(sue.pay)
```

□□□□□□□□□□□□□□□□□□□

□□□

```
# Add methods to encapsulate operations for maintainability

class Person:
    def __init__(self, name, job=None, pay=0):
        self.name = name
        self.job = job
        self.pay = pay
    def lastName(self):                          # Behavior methods
        return self.name.split()[-1]             # self is implied subject
    def giveRaise(self, percent):
        self.pay = int(self.pay * (1 + percent))         # Must change here only

if __name__ == '__main__':
    bob = Person('Bob Smith')
    sue = Person('Sue Jones', job='dev', pay=100000)
    print(bob.name, bob.pay)
    print(sue.name, sue.pay)
    print(bob.lastName(), sue.lastName())        # Use the new methods
    sue.giveRaise(.10)                           # instead of hardcoding
    print(sue.pay)
```

□□□

```
Bob Smith 0
Sue Jones 100000
Smith Jones
110000
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□sue□pay□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□

□□3□□□□□□

□□□□□□

□□□□□□□□□__init__:□□□__str__□□□□□□□□□□□□□□□□□□□□□□□□□__str__□□□□□□□□□□□□□□□□□□□

```
        self.name = name
        self.job = job
        self.pay = pay
    def lastName(self):
        return self.name.split()[-1]
    def giveRaise(self, percent):
        self.pay = int(self.pay * (1 + percent))
    def __str__(self):                                  # Added method
        return '[Person: %s, %s]' % (self.name, self.pay)   # String to print

if __name__ == '__main__':
    bob = Person('Bob Smith')
    sue = Person('Sue Jones', job='dev', pay=100000)
    print(bob)
    print(sue)
    print(bob.lastName(), sue.lastName())
    sue.giveRaise(.10)
    print(sue)
```

□□□□□□□□□□□□□□__str__□□□□□□%□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□__str__□□□□□□□□□□□__repr__□□□□□□□□□□
□□__str__□□
□□□□□

□□4□□□□□□□□□

□□□□□□□□□□□□□Python□□□□□OOP□□□□□□□□□□□□□

□□□□□□□□□...□□□□Manager□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□Manager□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□Manager□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□giveRaise□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

```
# Add customization of one behavior in a subclass

class Person:
    def __init__(self, name, job=None, pay=0):
        self.name = name
        self.job = job
        self.pay = pay
    def lastName(self):
        return self.name.split()[-1]
    def giveRaise(self, percent):
        self.pay = int(self.pay * (1 + percent))
    def __str__(self):
        return '[Person: %s, %s]' % (self.name, self.pay)

class Manager(Person):
    def giveRaise(self, percent, bonus=.10):              # Redefine at this level
        Person.giveRaise(self, percent + bonus)           # Call Person's version

if __name__ == '__main__':
    bob = Person('Bob Smith')
    sue = Person('Sue Jones', job='dev', pay=100000)
    print(bob)
    print(sue)
    print(bob.lastName(), sue.lastName())
    sue.giveRaise(.10)
    print(sue)
    tom = Manager('Tom Jones', 'mgr', 50000)              # Make a Manager: __init__
    tom.giveRaise(.10)                                    # Runs custom version
    print(tom.lastName())                                 # Runs inherited method
    print(tom)                                            # Runs inherited __str__
```

□□□□□□□Manager□□□□□□□□□□□self□□□□□□□□□□Manager□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
[Person: Bob Smith, 0]
[Person: Sue Jones, 100000]
Smith Jones
[Person: Sue Jones, 110000]
Jones
[Person: Tom Jones, 60000]
```

□□□□□□□□□□□□bob□sue□□□□□□□□□Manager tom□□10%□□□□□□□□□□□20%□□□pay□$50K□□□□$60K□□□□Manager□□□□giveRaise□□□□□□□□□□□□□□□□□□□□□□□□□□□□tom□□□□□□□□□□□□□□□□Person□__str__□□□□□□□□□□□□□Manager□□□□lastName□□__init__□□□□□□□□□□□□□□□□“□□”□Person□□□

□□□□□

□□□□

□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□giveRaise□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

OOP□□□□

□□□□

□□□

□□5□□□□□□□

□□□□□□□

```
# Add customization of constructor in a subclass

class Person:
    def __init__(self, name, job=None, pay=0):
        self.name = name
        self.job = job
        self.pay = pay
    def lastName(self):
        return self.name.split()[-1]
    def giveRaise(self, percent):
        self.pay = int(self.pay * (1 + percent))
    def __str__(self):
        return '[Person: %s, %s]' % (self.name, self.pay)

class Manager(Person):
    def __init__(self, name, pay):                    # Redefine constructor
        Person.__init__(self, name, 'mgr', pay)       # Run original with 'mgr'
    def giveRaise(self, percent, bonus=.10):
        Person.giveRaise(self, percent + bonus)


 if __name__ == '__main__':
    bob = Person('Bob Smith')
    sue = Person('Sue Jones', job='dev', pay=100000)
    print(bob)
    print(sue)
    print(bob.lastName(), sue.lastName())
    sue.giveRaise(.10)
    print(sue)
    tom = Manager('Tom Jones', 50000)                 # Job name not needed:
    tom.giveRaise(.10)                                # Implied/set by class
    print(tom.lastName())
    print(tom)
```

□□□□□□□□□□□□□□giveRaise□□□□□□□□□□□__init__□□□□——□□□□□□□□□□□□□self□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

·□□□□——□□□□□□□□□□□□□□□□□□□□

·□□□□□□□——□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□3□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□Python□□OOP□□□□□□□□□□□□□□□□□□

□□□□□□□Manager□□□□□□□

```
# Aggregate embedded objects into a composite

...
bob = Person(...)
sue = Person(...)
tom = Manager(...)

class Department:
    def __init__(self, *args):
        self.members = list(args)
    def addMember(self, person):
        self.members.append(person)
    def giveRaises(self, percent):
        for person in self.members:
            person.giveRaise(percent)
    def showAll(self):
        for person in self.members:
            print(person)

development = Department(bob, sue)          # Embed objects in a composite
development.addMember(tom)
development.giveRaises(.10)                 # Runs embedded objects' giveRaise
development.showAll()                       # Runs embedded objects' __str__s
```

□□□□□□□□□□□□□□□□□□□□□□□——Department□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Person□Manager□□□□□□□□□□□□□□□□□□□

□Python 3.0□□□□□□□□Python 2.6□□□□□□□□

□□6□□□□□□□

□□□□□□□

□□□

```
>>> from person import Person
>>> bob = Person('Bob Smith')
>>> print(bob)                                  # Show bob's __str__
[Person: Bob Smith, 0]

>>> bob.__class__                               # Show bob's class and its name
<class 'person.Person'>
>>> bob.__class__.__name__
'Person'

>>> list(bob.__dict__.keys())                   # Attributes are really dict keys
['pay', 'job', 'name']                          # Use list to force list in 3.0

>>> for key in bob.__dict__:
        print(key, '=>', bob.__dict__[key])     # Index manually

pay => 0
job => None
name => Bob Smith

>>> for key in bob.__dict__:
        print(key, '=>', getattr(bob, key))     # obj.attr, but attr is a var

pay => 0
job => None
name => Bob Smith
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__slots__□□□□□□□□□□□□□__dict__□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

```
class AttrDisplay:
    """
    Provides an inheritable print overload method that displays
    instances with their class names and a name=value pair for
    each attribute stored on the instance itself (but not attrs
    inherited from its classes). Can be mixed into any class,
    and will work on any instance.
    """
    def gatherAttrs(self):
        attrs = []
        for key in sorted(self.__dict__):
            attrs.append('%s=%s' % (key, getattr(self, key)))
        return ', '.join(attrs)
    def __str__(self):
        return '[%s: %s]' % (self.__class__.__name__, self.gatherAttrs())

if __name__ == '__main__':
    class TopTest(AttrDisplay):
        count = 0
        def __init__(self):
            self.attr1 = TopTest.count
            self.attr2 = TopTest.count+1
            TopTest.count += 2
    class SubTest(TopTest):
          pass

    X, Y = TopTest(), SubTest()
    print(X)                            # Show all instance attrs
    print(Y)                            # Show lowest class name
```

□□□

```
C:\misc> classtools.py
[TopTest: attr1=0, attr2=1]
[SubTest: attr1=2, attr2=3]
```

□□□□□□□□□□

□□□□□□□□□classtools□□□self□□□□□□□□□

```
>>> from person import Person
>>> bob = Person('Bob Smith')

# In Python 2.6:

>>> bob.__dict__.keys()                    # Instance attrs only
['pay', 'job', 'name']

>>> dir(bob)                               # + inherited attrs in classes
['__doc__', '__init__', '__module__', '__str__', 'giveRaise', 'job',
'lastName', 'name', 'pay']

# In Python 3.0:

>>> list(bob.__dict__.keys())              # 3.0 keys is a view, not a list
['pay', 'job', 'name']

>>> dir(bob)                               # 3.0 includes class type methods
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__',
...more lines omitted...
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'giveRaise', 'job', 'lastName', 'name', 'pay']
```

□□□□□□Python 2.6□Python 3.0□□□□□□□□Python 3.0□dict.keys□□□□□□□□□Python 3.0□dir□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□dir□□□□□□□□□□□□“□□”□□□Python 3.0□dir□□□□□□□□__X__□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□□□□classtools□□□□AttrDisplay□□□□□□□□□□□□□□□

□□□

□□□□□□

□□□□□

```
        return self.name.split()[-1]
    def giveRaise(self, percent):                    # Percent must be 0..1
        self.pay = int(self.pay * (1 + percent))

class Manager(Person):
    """
    A customized Person with special requirements
    """
    def __init__(self, name, pay):
        Person.__init__(self, name, 'mgr', pay)
    def giveRaise(self, percent, bonus=.10):
        Person.giveRaise(self, percent + bonus)

if __name__ == '__main__':
    bob = Person('Bob Smith')
    sue = Person('Sue Jones', job='dev', pay=100000)
    print(bob)
    print(sue)
    print(bob.lastName(), sue.lastName())
    sue.giveRaise(.10)
    print(sue)
    tom = Manager('Tom Jones', 50000)
    tom.giveRaise(.10)
    print(tom.lastName())
    print(tom)
```

□□□

```
C:\misc> person.py
[Person: job=None, name=Bob Smith, pay=0]
[Person: job=dev, name=Sue Jones, pay=100000]
Smith Jones
[Person: job=dev, name=Sue Jones, pay=110000]
Jones
[Manager: job=mgr, name=Tom Jones, pay=60000]
```

□□□

□□7□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

shelve

□□□□□□□□□□□□Python□□□□□□□□□□□

□□□□9□□□□□□□□□□□□

□□□□□□□□□□□□□□□shelve□□□□□□□□□□□□□□□

□shelve□□□□□□□□

Pickle□shelve□□□□□□□□

```
# File makedb.py: store Person objects on a shelve database

from person import Person, Manager              # Load our classes
bob = Person('Bob Smith')                       # Re-create objects to be stored
sue = Person('Sue Jones', job='dev', pay=100000)
tom = Manager('Tom Jones', 50000)

import shelve
db = shelve.open('persondb')                    # Filename where objects are stored
for object in (bob, sue, tom):                  # Use object's name attr as key
    db[object.name] = object                    # Store object on shelve by key
db.close()                                      # Close after making changes
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□shelve□□□□□□□□□□□□□□□shelve□□□□□□□□□□□□□□□□□□□

□□□□□shelve

□□□□

□□□

□□□

□□shelve□□□□

□□□□

```
c:\misc> updatedb.py
Bob Smith       => [Person: job=None, name=Bob Smith, pay=0]
Sue Jones       => [Person: job=dev, name=Sue Jones, pay=100000]
Tom Jones       => [Manager: job=mgr, name=Tom Jones, pay=50000]

c:\misc> updatedb.py
Bob Smith       => [Person: job=None, name=Bob Smith, pay=0]
Sue Jones       => [Person: job=dev, name=Sue Jones, pay=110000]
Tom Jones       => [Manager: job=mgr, name=Tom Jones, pay=50000]

c:\misc> updatedb.py
Bob Smith       => [Person: job=None, name=Bob Smith, pay=0]
Sue Jones       => [Person: job=dev, name=Sue Jones, pay=121000]
Tom Jones       => [Manager: job=mgr, name=Tom Jones, pay=50000]

c:\misc> updatedb.py
Bob Smith       => [Person: job=None, name=Bob Smith, pay=0]
Sue Jones       => [Person: job=dev, name=Sue Jones, pay=133100]
Tom Jones       => [Manager: job=mgr, name=Tom Jones, pay=50000]
```

□□□□□□□□□□□□□□□□Python□□□□□□□shelve□pickle□□□□□□□□□□□□□□□□

pickle□shelve□□□□□□□□□□□□□□□□Python□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□Python□OOP□□□□□□□□□□□□□□□□□□□

□□GUI□□□□□□□□Web□□□□□□□□□□□□□□□□□□Web□□□□□□□□□□□□□□□□GUI□□□□□□□Web□□□□□Python□□□□□CGI□□□□□□□□□□□□□□□Django□TurboGears□Pylons□web2Py□Zope□Google's App Engine□□□□□□□□□□Web□□□□□□□□□Web□□□□□□□□□□□□□□□□□□shelve□pickle□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Web□□□□□□□□□□□□□□□□□□□□□□HTML□□□□□□□□□□□□□□□□□□□□□□□□□□□API□□□□□□

Web□□

□□Web□□□□□□□□□□□□Web□□□□□□□□□□□□□□□□□□□

□□□□□□MySQL□□□□SQL□□□□□□

□□□□

□□Python□□OOP□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□shelve□□□□Manager□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□shelve□□□□Person□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□Manager□□□□□□classtools□□□AttrDisplay□□□__str__□□□□□Manager□□□□□□□□□□□□□□□□□□□□□□□□□□□□Person□□□□□□□□□□□__str__□□□□□□□□□□□□□□AttrDisplay□□□□□□□□□□□□□□□□□□□

5.□□□

第28章　面向对象编程

面向对象编程是一种程序设计思想。我们在Python OOP中讲过，面向对象编程把对象作为程序的基本单元，一个对象包含了数据和操作数据的函数。面向过程的程序设计把计算机程序视为一系列的命令集合，即一组函数的顺序执行。为了简化程序设计，面向过程把函数继续切分为子函数，即把大块函数通过切割成小块函数来降低系统的复杂度。

而面向对象的程序设计把计算机程序视为一组对象的集合，而每个对象都可以接收其他对象发过来的消息，并处理这些消息，计算机程序的执行就是一系列消息在各个对象之间传递。

class定义

在Python class中定义的方法其实就是OOP编程中的成员函数，只是我们在调用成员函数时，可以省略掉class的方法参数，这是Python的OOP编程和C++等其他语言不同的地方。在Python中class内部定义的方法def第一个class参数——这个参数就是实例本身，我们习惯性地把def第一个class参数命名为Python的习惯用法class参数名称为self，self可以帮我们绑定实例的属性。

小结

class□□□

□□

□□□

```
>>> class SharedData:
...     spam = 42                    # Generates a class data attribute
...
>>> x = SharedData()                 # Make two instances
>>> y = SharedData()
>>> x.spam, y.spam                   # They inherit and share 'spam'
(42, 42)
```

□□□□□□□□□spam□□class□□□□□□□□□

□□□

□□□□□□□□□□□□□□□□x.data□self.data□□□□□□□□□□□□□□□□

□□

□□□

C++□□□□□□□Python□self□□□C++□this□□□□□□□□□□Python□□self□□□□□□□□□□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□printer□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□x□□text□□□□□□□□□□□□□□□□□'instance call'□□□□□□□□ Python□□□□□□□□□□□□self□□□□□□□□□□□□□□□printer□□□□□self□□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

□□□□□□□□□□□

□□□31□□□□□□□□□□Python 2.2□□□□□□□□□□□□□□□□□□□

□□

□class□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□

□Python□□□□□□□□□□□□□□□□□□□

□ 28-1 □□□□

□□□□□□□□

□□□□□□□

```
>>> x = Super()                    # Make a Super instance
>>> x.method()                     # Runs Super.method
in Super.method

>>> x = Sub()                      # Make a Sub instance
>>> x.method()                     # Runs Sub.method, calls Super.method
starting Sub.method
in Super.method
ending Sub.method
```

□□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□specialize.py□□□□□□□□□□□□□□□□□□□□□□□□□

Super

□□□□method□□□□□□□□□□□□□□□□□□□delegate□

Inheritor

□□□□□□□□□□□□□□□□□□□□Super□□□□□□□□□□

Replacer

□□□□□□□□□Super□method□

Extender

□□□□□□□□method□□□□□□Super□method□

Provider

□□Super□delegate□□□□□□action□□□

□□□□□□□□□□□□□□□□□□□

```
class Super:
    def method(self):
        print('in Super.method')                    # Default behavior
    def delegate(self):
        self.action()                               # Expected to be defined

class Inheritor(Super):                             # Inherit method verbatim
    pass

class Replacer(Super):                              # Replace method completely
    def method(self):
        print('in Replacer.method')

class Extender(Super):                              # Extend method behavior
    def method(self):
        print('starting Extender.method')
        Super.method(self)
        print('ending Extender.method')

class Provider(Super):                              # Fill in a required method
    def action(self):
        print('in Provider.action')

if __name__ == '__main__':
    for klass in (Inheritor, Replacer, Extender):
        print('\n' + klass.__name__ + '...')
        klass().method()
```

```
print('\nProvider...')
x = Provider()
x.delegate()
```

□□□for□□□□□□□□□□□□□□□□□□□

```
% python specialize.py

Inheritor...
in Super.method

Replacer...
in Replacer.method

Extender...
starting Extender.method
in Super.method
ending Extender.method

Provider...
in Provider.action
```

□□□□

□□□□□□□□□□Provider□□□□□□□□□□□□□Provider□□□□delegate□□□□□□□□□□□□□□□□□□□□□

1.□□□x.delegate□□□□□□Python□□□Provider□□□□□□□□□□□□□□□Super□□□delegate□□□□□□□x□□□□□□□□□□□□□□□□self□□□

2.□Super.delegate□□□□self.action□□self□□□□□□□□□□□□□□□□□□□□□self□□□Provider□□□□Provider□□□□□□□action□□□

□□“□□”□□□□□□□□□□OOP□□□□□□□□□□□delegate□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□□□32□□□33□□□assert□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□31□□□□□□□8□□□□□□□□□□□□□□□□B□□
□□□□□□□□□OOP□□

Python 2.6□Python 3.0□□□□□

□Python 2.6□Python 3.0□□□□□□□□□□□□□□□“□□□□”□□□□□□□□□□□□□□□

```
>>> from abc import ABCMeta, abstractmethod
>>>
>>> class Super(metaclass=ABCMeta):
...     def delegate(self):
...         self.action()
...     @abstractmethod
...     def action(self):
...         pass
...
>>> X = Super()
TypeError: Can't instantiate abstract class Super with abstract methods action

>>> class Sub(Super): pass
...
>>> X = Sub()
TypeError: Can't instantiate abstract class Sub with abstract methods action

>>> class Sub(Super):
...     def action(self): print('spam')
...
>>> X = Sub()
>>> X.delegate()
spam
```

□□□

□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

Python□□□□□“□”□□□□□□□□□

□□□□□□□□□

□□□

```
# otherfile.py

import manynames

X = 66
print(X)                        # 66: the global here
print(manynames.X)              # 11: globals become attributes after imports

manynames.f()                   # 11: manynames's X, not the one here!
manynames.g()                   # 22: local in other file's function

print(manynames.C.X)            # 33: attribute of class in other module
I = manynames.C()
print(I.X)                      # 33: still from class here
I.m()
print(I.X)                      # 55: now from instance!
```

□□□manynames.f()□□□□□manynames□□X□□□□□□□□□□□□□□□□X□□□□□□□□□□□□□□□□

```
def g2():
    global X
    X = 22                              # Change global in module

def h1():
    X = 33                              # Local in function
    def nested():
    print(X)                            # Reference local in enclosing scope

def h2():
    X = 33                              # Local in function
    def nested():
        nonlocal X                      # Python 3.0 statement
        X = 44                          # Change local in enclosing scope
```

□□□

□□□□□□

□19□□□__dict__□□□□□□□□□□□□□□□□□□□

```
>>> X = sub()
>>> X.__dict__                          # Instance namespace dict
{}

>>> X.__class__                         # Class of instance
<class '__main__.sub'>

>>> sub.__bases__                       # Superclasses of class
(<class '__main__.super'>,)

>>> super.__bases__                     # () empty tuple in Python 2.6
(<class 'object'>,)
```

□□□self□□□

□□□Y□□□□□□□□□□□□□□□□□□□X□□□□□□□□□□□□□□□□□□□Y□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□Python 2.2□□dir□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□object□□□□□□□[1]□

```
>>> X.__dict__, Y.__dict__
({'data1': 'spam', 'data3': 'ham', 'data2': 'eggs'}, {})
>>> list(X.__dict__.keys())                                    # Need list in 3.0
['data1', 'data3', 'data2']

# In Python 2.6:

>>>> dir(X)
['__doc__', '__module__', 'data1', 'data2', 'data3', 'hello', 'hola']
>>> dir(sub)
['__doc__', '__module__', 'hello', 'hola']
>>> dir(super)
['__doc__', '__module__', 'hello']

# In Python 3.0:

>>> dir(X)
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__',
...more omitted...
'data1', 'data2', 'data3', 'hello', 'hola']

>>> dir(sub)
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__',
...more omitted...
'hello', 'hola']

>>> dir(super)
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__',
...more omitted...
'hello'
]
```

□□□□□□□□□□□□□□□□

Python 3.0□□□□□□□□□dir□□□□□□□□□

□□□□□□

□□□□□□“□□□□□□□□□□__class__□__bases__”□□□□□□□□□□

□□□

```
C:\misc> c:\python30\python classtree.py
Tree of <__main__.B object at 0x02810650>
...B
......A
.........object
Tree of <__main__.F object at 0x02810650>
...F
......D
.........B
............A
..............object
.........C
............A
..............object
......E
.........object
```

□□□

```
C:\misc> c:\python30\python
>>> class Emp: pass
...
>>> class Person(Emp): pass
>>> bob = Person()

>>> import classtree
>>> classtree.instancetree(bob)
Tree of <__main__.Person object at 0x028203B0>
...Person
......Emp
.........object
```

□□□

□□□□□□□

□□□□□□

```
>>> import docstr
>>> docstr.__doc__
'I am: docstr.__doc__'

>>> docstr.func.__doc__
'I am: docstr.func.__doc__'

>>> docstr.spam.__doc__
'I am: spam.__doc__ or docstr.spam.__doc__'

>>> docstr.spam.method.__doc__
'I am: spam.method.__doc__ or self.method.__doc__'
```

□15□□□□PyDoc□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□object□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□31□□□

```
>>> help(docstr)
Help on module docstr:

NAME
    docstr - I am: docstr.__doc__

FILE
    c:\misc\docstr.py

CLASSES
    spam

    class spam
    | I am: spam.__doc__ or docstr.spam.__doc__
    |
    | Methods defined here:
    |
    | method(self, arg)
    | I am: spam.method.__doc__ or self.method.__doc__

FUNCTIONS
    func(args)
        I am: docstr.func.__doc__
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□#□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

□□□□

□□□Python□□□OOP□□□

□□□□

1.□□□□□□□□□

2.□□□□□□□□□□□□class□□□□□□□□□□□□□□

3.□□□□□□□□□□□□□□□□□□□□__init__□□□

4.□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□

□29□　□□□□□

□□□□□

□□□

□□□□□□□□□□

□□□29-1□□□__add__□__radd__□__iadd__□□□□□□Python□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□

表29-1：常见运算符重载方法

方法	重载	调用
__init__	构造函数	对象建立：X = Class(args)
__del__	析构函数	X对象收回
__add__	运算符+	如果没有_iadd_，X + Y，X += Y
__or__	运算符|（位OR）	如果没有_ior_，X | Y，X |= Y
__repr__ ,__str__	打印、转换	print (X)、 repr(X)、 str(X)
__call__	函数调用	X(*args,**kargs)
__getattr__	点号运算	X.undefined
__setattr__	属性赋值语句	X.any = value
__delattr__	属性删除	del X.any
__getattribute__	属性获取	X.any

表29-1：常见运算符重载方法（续）

方法	重载	调用
__getitem__	索引运算	X[key],X[i:j]，没__iter__时的for循环和其他迭代器
__setitem__	索引赋值语句	X[key] = value, X[i:j] = sequence
__delitem__	索引和分片删除	del X[key], del X[i:j]
__len__	长度	len(X)，如果没有__bool__，真值测试
__bool__	布尔测试	bool(X)，真测试（在Python 2.6中叫做__nonzero__）
__lt__,__gt__, __le__,__ge__, __eq__,__ne__	特定的比较	X < Y, X > Y, X <= Y, X >=Y, X== Y, X != Y(或者在Python 2.6中只有__cmp__)
__radd__	右侧加法	Other + X
__iadd__	实地（增强的）加法	X += Y (or else __add__)
__iter__, __next__	迭代环境	I=iter(X), next(I); for loops, in if no __contains__, all comprehensions, map(F,X), 其他(__next__在Python 2.6中称为next)
__contains__	成员关系测试	item in X (任何可迭代的)
__index__	整数值	hex(X), bin(X), oct(X), O[X], O[X:]（替代Python 2中的__oct__、__hex__）
__enter__, __exit__	环境管理器（参见第33章）	with obj as var:
__get__, __set__, __delete__	描述符属性（参见第37章）	X.attr, X.attr = value, del X.attr
__new__	创建（参见第39章）	在__init__之前创建对象

□□□

□□□

□□□□□□__getitem__□__setitem__

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__getitem__□□□□X□□□X[i]□□□□□□□□□□□Python□□□□□□□□□□□__getitem__□□□□□□□□□□□□X□□□□□□□□□□□□□□□□□□□

```
>>> L = [5, 6, 7, 8, 9]
>>> L[2:4]                              # Slice with slice syntax
[7, 8]
>>> L[1:]
[6, 7, 8, 9]
>>> L[:-1]
[5, 6, 7, 8]
>>> L[::2]
[5, 7, 9]
```

□□□

```
>>> class Indexer:
...     data = [5, 6, 7, 8, 9]
...     def __getitem__(self, index):          # Called for index or slice
...         print('getitem:', index)
...         return self.data[index]            # Perform index or slice
...
>>> X = Indexer()
>>> X[0]                                       # Indexing sends __getitem__ an integer
getitem: 0
5
>>> X[1]
getitem: 1
6
>>> X[-1]
getitem: -1
9
```

□□□

```
...
self.data[index] = value                    # Assign index or slice
```

□□□□__getitem__□□□□

```
>>> class C:
...     def __index__(self):
...         return 255
...
>>> X = C()
>>> hex(X)                          # Integer value
'0xff'
>>> bin(X)
'0b11111111'
>>> oct(X)
'0o377'
```

□□□□□□□□□□□□□__getitem__□□□□□□□□□□□□□

□□□□□__getitem__

□□for□□□□□□□□0□□□□□□□□□□□□□□□□□□□

```
>>> 'p' in X                          # All call __getitem__ too
True

>>> [c for c in X]                    # List comprehension
['S', 'p', 'a', 'm']

>>> list(map(str.upper, X))           # map calls (use list() in 3.0)
['S', 'P', 'A', 'M']

>>> (a, b, c, d) = X                  # Sequence assignments
>>> a, c, d
('S', 'a', 'm')

>>> list(X), tuple(X), ''.join(X)
(['S', 'p', 'a', 'm'], ('S', 'p', 'a', 'm'), 'Spam')

>>> X
<__main__.stepper object at 0x00A8D5D0>
```

□□□

□□□□□□□__iter__□__next__

□□□□□□□□__getitem__□□□□□□□□

□__iter__□□□□□□□□□□□□□14□□□20□□□□□□□□□□□□□

```
>>> X = Squares(1, 5)                    # Iterate manually: what loops do
>>> I = iter(X)                          # iter calls __iter__
>>> next(I)                              # next calls __next__
1
>>> next(I)
4
...more omitted...
>>> next(I)
25
>>> next(I)                              # Can catch this in try statement
StopIteration
```

__getitem__□□□□□□□□□□□□□□□□□□□□for□□□□□0□□□□□□□□□□□□□□□□□□□

```
>>> X = Squares(1, 5)
>>> [n for n in X]                          # Exhausts items
[1, 4, 9, 16, 25]
>>> [n for n in X]                          # Now it's empty
[]
>>> [n for n in Squares(1, 5)]              # Make a new iterator object
[1, 4, 9, 16, 25]
>>> list(Squares(1, 3))
[1, 4, 9]
```

□□□□□□□□□□□□□□□□20□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> def gsquares(start, stop):
...     for i in range(start, stop+1):
...         yield i ** 2
...
>>> for i in gsquares(1, 5):                # or: (x ** 2 for x in range(1, 5))
...     print(i, end=' ')
...
1 4 9 16 25
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

```
class SkipIterator:
    def __init__(self, wrapped):
        self.wrapped = wrapped                  # Iterator state information
        self.offset = 0
    def __next__(self):
        if self.offset >= len(self.wrapped):    # Terminate iterations
            raise StopIteration
        else:
            item = self.wrapped[self.offset]    # else return and skip
            self.offset += 2
            return item

class SkipObject:
    def __init__(self, wrapped):                # Save item to be used
        self.wrapped = wrapped
    def __iter__(self):
        return SkipIterator(self.wrapped)       # New iterator each time

if __name__ == '__main__':
    alpha = 'abcdef'
    skipper = SkipObject(alpha)                 # Make container object
    I = iter(skipper)                           # Make an iterator on it
    print(next(I), next(I), next(I))            # Visit offsets 0, 2, 4

    for x in skipper:                           # for calls __iter__ automatically
        for y in skipper:                       # Nested fors call __iter__ again each time
            print(x + y, end=' ')               # Each iterator has its own state, offset
```

□□□

```
>>> S = 'abcdef'
>>> for x in S[::2]:
...     for y in S[::2]:                          # New objects on each iteration
...         print(x + y, end=' ')
...
```

```
aa ac ae ca cc ce ea ec ee
```

□□□

□□□□□__contains__□__iter__□__getitem__

□□□□

```
class Iters:
    def __init__(self, value):
        self.data = value
    def __getitem__(self, i):                    # Fallback for iteration
        print('get[%s]:' % i, end='')            # Also for index, slice
        return self.data[i]
    def __iter__(self):                          # Preferred for iteration
        print('iter=> ', end='')                 # Allows only 1 active iterator
        self.ix = 0
        return self
    def __next__(self):
        print('next:', end='')
        if self.ix == len(self.data): raise StopIteration
        item = self.data[self.ix]
        self.ix += 1
        return item
    def __contains__(self, x):                   # Preferred for 'in'
        print('contains: ', end='')
        return x in self.data

X = Iters([1, 2, 3, 4, 5])                       # Make instance
print(3 in X)                                    # Membership
for i in X:                                      # For loops
print(i, end=' | ')

print()
print([i ** 2 for i in X])                       # Other iteration contexts
print( list(map(bin, X)) )

I = iter(X)                                      # Manual iteration (what other contexts do)
while True:
    try:
        print(next(I), end=' @ ')
    except StopIteration:
        break
```

□□□□□□□□□□□□□□□□□□□——□□□__contains__□□□□□□□□□□

__iter__□□□□□□□□□□□__next__□□□□□□□□__getitem__□□□□□□

```
contains: True
iter=> next:1 | next:2 | next:3 | next:4 | next:5 | next:
iter=> next:next:next:next:next:next:[1, 4, 9, 16, 25]
iter=> next:next:next:next:next:next:['0b1', '0b10', '0b11', '0b100', '0b101']
iter=> next:1 @ next:2 @ next:3 @ next:4 @ next:5 @ next:
```

□□□□□□□□□□□□__contains__□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□__iter__□

```
iter=> next:next:next:True
iter=> next:1 | next:2 | next:3 | next:4 | next:5 | next:
iter=> next:next:next:next:next:next:[1, 4, 9, 16, 25]
iter=> next:next:next:next:next:next:['0b1', '0b10', '0b11', '0b100', '0b101']
iter=> next:1 @ next:2 @ next:3 @ next:4 @ next:5 @ next:
```

□□□□□__contains__□__iter__□□□□□□□□□□□□□——□□__getitem__□□□□□□□□□□□□□□□□

```
>>> X = Iters('spam')                    # Indexing
>>> X[0]                                 # __getitem__(0)
get[0]:'s'

>>> 'spam'[1:]                           # Slice syntax
'pam'
>>> 'spam'[slice(1, None)]               # Slice object
'pam'

>>> X[1:]                                # __getitem__(slice(..))
get[slice(1, None, None)]:'pam'
>>> X[:-1]
get[slice(None, ?1, None)]:'spa'
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□__iter__□□__contains__□□□□□□□□□□□□□□□□□□□□□□

□□□□□__getattr__□__setattr__

__getattr__□□□□

□□□□□□□□□□□□□□□□□□□□__getattr__□□□□□□AttributeError□□□□□Python□□□□□□□□□□□□□□□□X.name□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□

·__getattribute__□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

□□□□□□□38□□□□□□□

__repr__和__str__显示方法的比较

在前面的例子中，__init__和__add__方法创建并增加了类实例对象，但是当我们试图显示这些对象时，默认的__repr__显示的只是self.data的内容，而且这个显示不是很美观。要定制对象的显示，__repr__或__str__方法可以返回一个新的字符串，以便显示时使用。

基本的显示定制

```
>>> class adder:
...     def __init__(self, value=0):
...         self.data = value                    # Initialize data
...     def __add__(self, other):
...         self.data += other                   # Add other in-place (bad!)
...
>>> x = adder()                                  # Default displays
>>> print(x)
<__main__.adder object at 0x025D66B0>
>>> x
<__main__.adder object at 0x025D66B0>
```

创建定制的显示

```
>>> class addrepr(adder):                        # Inherit __init__, __add__
...     def __repr__(self):                      # Add string representation
...         return 'addrepr(%s)' % self.data     # Convert to as-code string
...
>>> x = addrepr(2)                                      # Runs __init__
>>> x + 1                                               # Runs __add__
>>> x                                                   # Runs __repr__
addrepr(3)
```

```
>>> print(x)                                    # Runs __repr__
addrepr(3)
>>> str(x), repr(x)                             # Runs __repr__ for both
('addrepr(3)', 'addrepr(3)')
```

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__repr__□□□□□□□□□□□□□□□□□□□

```
>>> class Printer:
...     def __init__(self, val):
...         self.val = val
...     def __str__(self):                    # Used for instance itself
...         return str(self.val)              # Convert to a string result
...
>>> objs = [Printer(2), Printer(3)]
>>> for x in objs: print(x)                   # __str__ run when instance printed
...                                           # But not when instance in a list!
2
3
>>> print(objs)
[<__main__.Printer object at 0x025D06F0>, <__main__.Printer object at ...more...
>>> objs
[<__main__.Printer object at 0x025D06F0>, <__main__.Printer object at ...more...
```

□□□

□□□□□□□□□□□__radd__□__iadd__

□□□□□□□□□□□□□□□□□__add__□□□□□□+□□□__radd__□□□□□□+□□□□□□□□□□□□□□□□□□□

□□□

□□□□□

```
>>> class Number:
...     def __init__(self, val):
...         self.val = val
...     def __iadd__(self, other):              # __iadd__ explicit: x += y
...         self.val += other                   # Usually returns self
...         return self
...
>>> x = Number(5)
>>> x += 1
>>> x += 1
>>> x.val
7
>>> class Number:
...     def __init__(self, val):
...         self.val = val
...     def __add__(self, other):               # __add__ fallback: x = (x + y)
...         return Number(self.val + other)     # Propagates class type
...
>>> x = Number(5)
>>> x += 1
>>> x += 1
>>> x.val
7
```

□□□□□

Call□□□□__call__

□□□□□□□□□__call__□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□__call__□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> class Callee:
...     def __call__(self, *pargs, **kargs):      # Intercept instance calls
...         print('Called:', pargs, kargs)        # Accept arbitrary arguments
...
>>> C = Callee()
>>> C(1, 2, 3)                                    # C is a callable object
Called: (1, 2, 3) {}
>>> C(1, 2, 3, x=4, y=5)
Called: (1, 2, 3) {'y': 5, 'x': 4}
```

□□□□□□□□□□18□□□□□□□□□□□□□□__call__□□□□□——□□□□□□□□□□□□□□□□□□□

```
X = C()
X(1, 2)                          # Omit defaults
X(1, 2, 3, 4)                    # Positionals
X(a=1, b=2, d=4)                 # Keywords
X(*[1, 2], **dict(c=3, d=4))     # Unpack arbitrary arguments
X(1, *(2,), c=3, **dict(d=4))    # Mixed modes
```

□□□□□□□□□□□□__call__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```
>>> class Prod:
...     def __init__(self, value):
...         self.value = value
...     def comp(self, other):
...         return self.value * other
...
>>> x = Prod(3)
>>> x.comp(3)
9
>>> x.comp(4)
12
```

□□□□□□□□□□API□□□□□□__call__□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□__call__□□□□□□□GUI□□□□□□□□□□□□□□□□□□□

```
cb3 = (lambda color='red': 'turn ' + color)      # Or: defaults
print(cb3())
```

□□□□□□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□

□□□__lt__□__gt__□□□□□

□□□29-1□□□□□□□□□□□□□□□□□□□□6□□□□□□□□□□□□□□□=□□=□==□!=□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·□□□□□□__add__/__radd__□□□□□□□□□□□□□□□□□□□

□Python 3.0□Python 2.6□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□

□□□□□__bool__□__len__

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□Python□□□□□__bool__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__len__□□□□□□□□□□□□□□□□□□□

```
>>> class Truth:
...     def __len__(self): return 0
...
>>> X = Truth()
>>> if not X: print('no!')
...
no!
```

□□□□□□□□□Python□□__bool__□□__len__□□□□□□□□

```
>>> class Truth:
...     def __bool__(self): return True          # 3.0 tries __bool__ first
...     def __len__(self): return 0              # 2.6 tries __len__ first
...
>>> X = Truth()
>>> if X: print('yes!')
...
yes!
```

□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> class Truth:
...     pass
...
>>> X = Truth()
>>> bool(X)
True
```

□□□□□□□□□□□□□□□□□□□

Python 2.6□□□□

Python 2.6□□□□□“□□□□□__bool__□__len__”□□□□□□□□□□ __nonzero__□□□__bool__□Python 3.0□Python 2.6□__nonzero__ □□□□□□□__bool__□□□□□□□□□□□□□□□□□Python 3.0□Python 2.6□□□ __len__□□□□□□

□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ __bool__□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□False□

```
C:\misc> c:\python30\python
>>> class C:
...     def __bool__(self):
...         print('in bool')
...         return False
...
>>> X = C()
>>> bool(X)
in bool
False
>>> if X: print(99)
...
in bool
```

□□Python 3.0□□□□□□□□□□□□□□□□□Python 2.6□□__bool__□□□□□□□□□□□□□

```
C:\misc> c:\python26\python
>>> class C:
...     def __bool__(self):
...         print('in bool')
...         return False
...
>>> X = C()
>>> bool(X)
True
>>> if X: print(99)
...
99
```

□Python 2.6□□□□□□□□□□__nonzero__□□□□□□□□□□__len__□□□□□□0□□

```
C:\misc> c:\python26\python
>>> class C:
...     def __nonzero__(self):
...         print('in nonzero')
...         return False
...
>>> X = C()
>>> bool(X)
in nonzero
False
>>> if X: print(99)
...
in nonzero
```

□□□□□□□__nonzero__□□Python 2.6□□□□□□□Python 3.0□□□□□□□

□□□□□□□__del__

□□□□□□□□□□□□__init__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__del__□□□□□□□□□destructor method□□□□□□□□□

```
>>> class Life:
...     def __init__(self, name='unknown'):
...         print('Hello', name)
...         self.name = name
...     def __del__(self):
...         print('Goodbye', self.name)
...
>>> brian = Life('Brian')
Hello Brian
>>> brian = 'loretta'
Goodbye Brian
```

□□□□□brian□□□□□□□□□□□□□□Life□□□□□□□□□□□□□□□□□□□

try/finally□□□

□□□□

□□□

□□□□

1.□□□

□□□□

1.□□□□□□□□□□□□□□__getitem__□__iter__□□□□□□□□□□□□□□

□30□　□□□□

□□□□□□□□□□□□□□□□□□□□□□Python□OOP□□□□□□□□□OOP□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□Python□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□def□□□□□□□□□□□□□□□□□□□

OOP□□□□“□□□”□□

□□□□

```
class Employee:
    def __init__(self, name, salary=0):
        self.name = name
        self.salary = salary
    def giveRaise(self, percent):
        self.salary = self.salary + (self.salary * percent)
    def work(self):
        print(self.name, "does stuff")
    def __repr__(self):
        return "<Employee: name=%s, salary=%s>" % (self.name, self.salary)

class Chef(Employee):
    def __init__(self, name):
        Employee.__init__(self, name, 50000)
    def work(self):
        print(self.name, "makes food")

class Server(Employee):
    def __init__(self, name):
        Employee.__init__(self, name, 40000)
    def work(self):
        print(self.name, "interfaces with customer")

class PizzaRobot(Chef):
    def __init__(self, name):
        Chef.__init__(self, name)
    def work(self):
        print(self.name, "makes pizza")


if __name__ == "__main__":
    bob = PizzaRobot('bob')                  # Make a robot named bob
    print(bob)                               # Run inherited __repr__
    bob.work()                               # Run type-specific action
    bob.giveRaise(0.20)                      # Give bob a 20% raise
    print(bob); print()

    for klass in Employee, Chef, Server, PizzaRobot:
        obj = klass(klass.__name__)
        obj.work()
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bob□□□□□□□□□□□□□□□□□□□□□□□

PizzaRobot□Chef□□Employee□□□□□□bob□□□

Employee.__repr__□□□□□□bob□□□□□□□Employee.giveRaise□□
□□□□□□□□□□□□□□□

```
C:\python\examples> python employees.py
<Employee: name=bob, salary=50000>
bob makes pizza
<Employee: name=bob, salary=60000.0>

Employee does stuff
Chef makes food
Server interfaces with customer
PizzaRobot makes pizza
```

□□□□□□□□□□□□□□□

OOP□□□□“□□□”□□

□□25□□□□

```
from employees import PizzaRobot, Server

class Customer:
    def __init__(self, name):
        self.name = name
    def order(self, server):
        print(self.name, "orders from", server)
    def pay(self, server):
        print(self.name, "pays for item to", server)

class Oven:
    def bake(self):
        print("oven bakes")

class PizzaShop:
    def __init__(self):
        self.server = Server('Pat')                    # Embed other objects
        self.chef = PizzaRobot('Bob')                  # A robot named bob
        self.oven = Oven()

    def order(self, name):
        customer = Customer(name)                      # Activate other objects
        customer.order(self.server)                    # Customer orders from server
        self.chef.work()
        self.oven.bake()
        customer.pay(self.server)

if __name__ == "__main__":
    scene = PizzaShop()                                # Make the composite
    scene.order('Homer')                               # Simulate Homer's order
    print('...')
    scene.order('Shaggy')                              # Simulate Shaggy's order
```

PizzaShop□□□

```
C:\python\examples> python pizzashop.py
Homer orders from <Employee: name=Pat, salary=40000>
Bob makes pizza
oven bakes
Homer pays for item to <Employee: name=Pat, salary=40000>
...
Shaggy orders from <Employee: name=Pat, salary=40000>


Bob makes pizza
oven bakes
Shaggy pays for item to <Employee: name=Pat, salary=40000>
```

□□□

```
class Processor:
    def __init__(self, reader, writer):
        self.reader = reader
        self.writer = writer
    def process(self):
        while 1:
            data = self.reader.readline()
            if not data: break
            data = self.converter(data)
            self.writer.write(data)
    def converter(self, data):
        assert False, 'converter must be defined'          # Or raise exception
```

□□□

□□□□□□□□□□□□□□□□□□□□stdout□□

```
C:\lp4e> type spam.txt
spam
Spam
SPAM!

C:\lp4e> python converters.py
SPAM
SPAM
SPAM!
```

□□□□□□□□□□□□□□□□□□□

```
C:\lp4e> python
>>> from converters import Uppercase
>>>
>>> class HTMLize:
...     def write(self, line):
...       print('<PRE>%s</PRE>' % line.rstrip())
...
>>> Uppercase(open('spam.txt'), HTMLize()).process()
<PRE>SPAM</PRE>
<PRE>SPAM</PRE>
<PRE>SPAM!</PRE>
```

□□□□□Processor□□□□□

□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□pickle□□□□□□□□□□□□□□□□

```
import shelve
object = someClass()
dbase = shelve.open('filename')
dbase['key'] = object                          # Save under key

import shelve
dbase = shelve.open('filename')
object = dbase['key']                          # Fetch it back later
```

□□□

OOP□□□□“□□”□□

□□□□□□□□□□□□□□□□□□□□□□delegation□□□□□□□□□□□□□□□□□□□

```
>>> from trace import wrapper
>>> x = wrapper([1,2,3])                  # Wrap a list
>>> x.append(4)                           # Delegate to list method
Trace: append
>>> x.wrapped                             # Print my member
[1, 2, 3, 4]

>>> x = wrapper({"a": 1, "b": 2})         # Wrap a dictionary
>>> x.keys()                              # Delegate to dictionary method
Trace: keys
['a', 'b']
```

□□□

□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
class C3(C1, C2): ...
I = C3()                                        # Only 1 X in I!
```

□□□□□□□□self.X□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□self.X□□□□□□□□□□□□□□□□□X□□□□□□□□I.X□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```
% python private.py
{'_C2__X': 99, '_C1__X': 88}
88
99
```

□□□

□□□

□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□29□□□□□__call__□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□Python□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□doit□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□Python 3.0□Python 2.6□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
>>> bound = x.double
>>> bound.__self__, bound.__func__
(<__main__.Number object at 0x0278F610>, <function double at 0x027A4ED0>)
>>> bound.__self__.base
2
>>> bound()                                  # Calls bound.__func__(bound.__self__, ...)
4
```

□□□□□□□□□□Python□□□□□□□□□□

```
15
>>> actions[-1](5)                          # Index, comprehensions, maps
15
>>> [act(5) for act in actions]
[25, 7, 15]
>>> list(map(lambda act: act(5), actions))
[25, 7, 15]
```

□□□

□□□

□□□

□□□□□“□□”□

□□□□

□□□□□□□

□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> class Spam:
...     def __init__(self):                    # No __repr__ or __str__
...         self.data1 = "food"
...
>>> X = Spam()
>>> print(X)                                   # Default: class, address
<__main__.Spam object at 0x00864818>           # Displays "instance" in Python 2.6
```

□□□□□□29□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__str__□__repr___□□□__repr___□□□□□□□□□□□□□□□□□□□

□__dict__□□□□□□

□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□

□□□□□□ListInstance□□□□27□□□□□□

```
>>> str(x)
'<Instance of Spam, address 40240880:\n\tname data1=food\n>'
>>> x                                          # The __repr__ still is a default
<__main__.Spam object at 0x026606F0>
```

ListInstance□□□

```
C:\misc> C:\python30\python testmixin.py
<Instance of Sub, address 40962576:
        name data1=spam
        name data2=eggs
        name data3=42
>
```

ListInstance□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□

□□□□_str__□□□□

□□□□□□□□□□□__X__□□□□□□□□□□□

```
C:\misc> c:\python30\python testmixin.py
<Instance of Sub, address 40831792:
        name _ListInherited__attrnames=<bound method Sub.__attrnames of <...more...>>
        name __class__=<>
        name __delattr__=<>
        name __dict__=<>
        name __doc__=<>
        name __eq__=<>
        ...more names omitted...
        name __repr__=<>
        name __setattr__=<>
        name __sizeof__=<>
        name __str__=<>
        name __subclasshook__=<>
        name __weakref__=<>
        name data1=spam
        name data2=eggs
        name data3=42
        name ham=<bound method Sub.ham of <__main__.Sub object at 0x026F0B30>>
        name spam=<bound method Sub.spam of <__main__.Sub object at 0x026F0B30>>
>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__str__□□□__repr__□□□□□□□□__repr__□□__repr__□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□__class__□□□□□□□□□□□□□__bases__□□□□□□□□□□□□□□□□__dicts__□

File lister.py, continued

```
class ListTree:
    """
    Mix-in that returns an __str__ trace of the entire class
    tree and all its objects' attrs at and above self;
    run by print(), str() returns constructed string;
    uses __X attr names to avoid impacting clients;
    uses generator expr to recurse to superclasses;
    uses str.format() to make substitutions clearer
    """
    def __str__(self):
        self.__visited = {}
        return '<Instance of {0}, address {1}:\n{2}{3}>'.format(
                            self.__class__.__name__,
                            id(self),
                            self.__attrnames(self, 0),
                            self.__listclass(self.__class__, 4))

    def __listclass(self, aClass, indent):
        dots = '.' * indent
        if aClass in self.__visited:
            return '\n{0}<Class {1}:, address {2}: (see above)>\n'.format(
                            dots,
                            aClass.__name__,
                            id(aClass))
        else:
            self.__visited[aClass] = True
            genabove = (self.__listclass(c, indent+4) for c in aClass.__bases__)
            return '\n{0}<Class {1}, address {2}:\n{3}{4}{5}>\n'.format(
                            dots,
                            aClass.__name__,
                            id(aClass),
                            self.__attrnames(aClass, indent),
                            ''.join(genabove),
                            dots)

    def __attrnames(self, obj, indent):
        spaces = ' ' * (indent + 4)
        result = ''
        for attr in sorted(obj.__dict__):
            if attr.startswith('__') and attr.endswith('__'):
                result += spaces + '{0}=<>\n'.format(attr)
            else:
                result += spaces + '{0}={1}\n'.format(attr, getattr(obj, attr))
        return result
```

□□□

```
C:\misc> c:\python26\python testmixin.py
<Instance of Sub, address 40728496:
    _ListTree__visited={}
    data1=spam
    data2=eggs
    data3=42

....<Class Sub, address 40701168:
        __doc__=<>
        __init__=<>
        __module__=<>
        spam=<unbound method Sub.spam>

........<Class Super, address 40701120:
            __doc__=<>
            __init__=<>
            __module__=<>
            ham=<unbound method Super.ham>
........>

........<Class ListTree, address 40700688:
            _ListTree__attrnames=<unbound method ListTree.__attrnames>
            _ListTree__listclass=<unbound method ListTree.__listclass>
            __doc__=<>
            __module__=<>
            __str__=<>
........>
....>
>
```

□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python testmixin.py
<Instance of Sub, address 40635216:
    _ListTree__visited={}
    data1=spam
    data2=eggs
```

```
    data3=42

....<Class Sub, address 40914752:
        __doc__=<>
        __init__=<>
        __module__=<>
        spam=<function spam at 0x026D53D8>

........<Class Super, address 40829952:
            __dict__=<>
            __doc__=<>
            __init__=<>
            __module__=<>
            __weakref__=<>
            ham=<function ham at 0x026D5228>

............<Class object, address 505114624:
                __class__=<>
                __delattr__=<>
                __doc__=<>
                __eq__=<>
                ...more omitted...
                __repr__=<>
                __setattr__=<>
                __sizeof__=<>
                __str__=<>
                __subclasshook__=<>
............>
........>

........<Class ListTree, address 40829496:
            _ListTree__attrnames=<function __attrnames at 0x026D5660>
            _ListTree__listclass=<function __listclass at 0x026D56A8>
            __dict__=<>
            __doc__=<>
            __module__=<>
            __str__=<>
            __weakref__=<>
............<Class object:, address 505114624: (see above)>
........>
....>
>
```

□□□

□□□□□□□□□□□□□__X__□□□□□□□□□□□□□

```
>>> from lister import ListTree
>>> from tkinter import Button                    # Both classes have a __str__
>>> class MyButton(ListTree, Button): pass        # ListTree first: use its __str__
...
>>> B = MyButton(text='spam')
>>> open('savetree.txt', 'w').write(str(B))       # Save to a file for later viewing
18247
>>> print(B)                                      # Print the display here
<Instance of MyButton, address 44355632:
    _ListTree__visited={}
    _name=44355632
    _tclCommands=[]
    ...much more omitted...
>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□GUI□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□slot□□□□__dict__□□□□□□□□□□□□

□□□□□□□□□□□□□

□□□□□□□□□□□□

□□

□□□

□□□

□□□□□□□□□□□

□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□□

□□□□

1.□□□□□□□□□

2.□□□□□□

3.□□□□□□

4.□□□□□□□□

5.□□□□□□□□□□□?

□□□□

1.□□□

□31□ □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□6□□□Python OOP□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□30□□□□□□□□□□□□□□Python 2.2□□□□□□□□□□□□□□□□□□□

```
class Set:
   def __init__(self, value = []):            # Constructor
       self.data = []                          # Manages a list
       self.concat(value)

   def intersect(self, other):                 # other is any sequence
       res = []                                # self is the subject
       for x in self.data:
           if x in other:                      # Pick common items
               res.append(x)
       return Set(res)                         # Return a new Set

   def union(self, other):                     # other is any sequence
       res = self.data[:]                      # Copy of my list
       for x in other:                         # Add items in other
           if not x in res:
               res.append(x)
       return Set(res)

   def concat(self, value):                    # value: list, Set...
       for x in value:                         # Removes duplicates
          if not x in self.data:
               self.data.append(x)

   def __len__(self):         return len(self.data)            # len(self)
   def __getitem__(self, key):return self.data[key]            # self[i]
   def __and__(self, other):  return self.intersect(other)     # self & other
   def __or__(self, other):   return self.union(other)         # self | other
   def __repr__(self):        return 'Set:' + repr(self.data)  # print()
```

□□

```
x = Set([1, 3, 5, 7])
print(x.union(Set([1, 4, 7])))          # prints Set:[1, 3, 5, 7, 4]
print(x | Set([1, 4, 6]))               # prints Set:[1, 3, 5, 7, 4, 6]
```

□□□□□□□Set□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□Python 2.2□□□□□□□

□□□□□

```
% python typesubclass.py
['a', 'b', 'c']
['a', 'b', 'c']
(indexing ['a', 'b', 'c'] at 1)
a
(indexing ['a', 'b', 'c'] at 3)
c
['a', 'b', 'c', 'spam']
['spam', 'c', 'b', 'a']
```

□□□□□

```
class Set(list):
    def __init__(self, value = []):            # Constructor
        list.__init__([])                       # Customizes list
        self.concat(value)                      # Copies mutable defaults

    def intersect(self, other):                 # other is any sequence
        res = []                                # self is the subject
        for x in self:
            if x in other:                      # Pick common items
               res.append(x)
        return Set(res)                         # Return a new Set

    def union(self, other):                     # other is any sequence
        res = Set(self)                         # Copy me and my list
        res.concat(other)
        return res

    def concat(self, value):                    # value: list, Set . . .
        for x in value:                         # Removes duplicates
            if not x in self:
                self.append(x)

    def __and__(self, other): return self.intersect(other)
    def __or__(self, other):  return self.union(other)
    def __repr__(self):       return 'Set:' + list.__repr__(self)

if __name__ == '__main__':
    x = Set([1,3,5,7])
    y = Set([2,1,4,5,6])
    print(x, y, len(x))
    print(x.intersect(y), y.union(x))
    print(x & y, x | y)
    x.reverse(); print(x)
```

□□□

```
% python setsubclass.py
Set:[1, 3, 5, 7] Set:[2, 1, 4, 5, 6] 4
Set:[1, 5] Set:[2, 1, 4, 5, 6, 3, 7]
Set:[1, 5] Set:[1, 3, 5, 7, 2, 4, 6]
Set:[7, 5, 3, 1]
```

Python□□□

□□□

□Python 2.2□□□□□□□□□□□□□“□□”□new-style□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□”□classic□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□Python 2.X□□□□□□□□□□□□□□□□

·□□Python 3.0□□□□□□□□□□□□□□□□“□□□”□□□□□□□□□□□□□□□□□object□□□□□□□□□□object□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□object□□□□

·□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□”object□□□□□□□□□□□□□□□□□□□

```
class newstyle(object):
    ...normal code...
```

□□□□□□□□□object□□□□□□□□□

□□□□□

□□□□

□□□□27□□□□

```
C:\misc> c:\python26\python
>>> class C: pass                          # Classic classes in 2.6
...
>>> I = C()
>>> type(I)                                # Instances are made from classes
<type 'instance'>
>>> I.__class__
<class __main__.C at 0x025085A0>

>>> type(C)                                # But classes are not the same as types
<type 'classobj'>
>>> C.__class__
AttributeError: class C has no attribute '__class__'

>>> type([1, 2, 3])
<type 'list'>
>>> type(list)
<type 'type'>
>>> list.__class__
<type 'type'>
```

□□□□□Python 2.6□□□□□

```
>>> type(C)                          # Classes are user-defined types
<type 'type'>
>>> C.__class__
<type 'type'>

>>> type([1, 2, 3])                  # Built-in types work the same way
<type 'list'>
>>> type(list)
<type 'type'>
>>> list.__class__
<type 'type'>
```

□□Python 3.0□□□

□□□

```
C:\misc> c:\python26\python
>>> class C: pass
...
>>> class D: pass
...
>>> c = C()
>>> d = D()
>>> type(c) == type(d)                    # 2.6: all instances are same type
True
>>> c.__class__ == d.__class__            # Must compare classes explicitly
False

>>> type(c), type(d)
(<type 'instance'>, <type 'instance'>)
>>> c.__class__, d.__class__
(<class __main__.C at 0x024585A0>, <class __main__.D at 0x024588D0>)
```

□□□□□□□□□□□□□□□□□Python 2.6□□□□□□Python 3.0□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python26\python
>>> class C(object): pass
...
>>> class D(object): pass
...
>>> c = C()
>>> d = D()
>>> type(c) == type(d)                    # 2.6 new-style: same as all in 3.0
False
```

```
>>> type(c), type(d)
(<class '__main__.C'>, <class '__main__.D'>)
>>> c.__class__, d.__class__
(<class '__main__.C'>, <class '__main__.D'>)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□

□□□□□

```
>>> class C(A):
        attr = 2

>>> class D(B, C):
        pass                                  # Tries A before C

>>> x = D()
>>> x.attr                                    # Searches x, D, B, A
1
```

□□□□□□A□□□□□□

□□□□□□□□□□□□□□□C□□□□□□□□□C□□□□□□□□□□

□□□□□□□□□□□□□□□C□□□□A□□□□□□□□□□□□C□□□□C□□□□□□□□□□□□□□□□□A□□□□□□□□C□□□

□□□□□□

□□□□□□□□□□□□□□□□□

```
>>> class A(object):
        attr = 1                            # New-style

>>> class B(A):
        pass

>>> class C(A):
        attr = 2

>>> class D(B, C):
        attr = B.attr                       # Choose A.attr, above

>>> x = D()
>>> x.attr                                  # Works like classic (default 2.6)
1
```

□□□

```
>>> class D(B, C): meth = C.meth                # Pick C's method: new-style (and 3.0)
>>> x = D()
>>> x.meth()
C.meth

>>> class D(B, C): meth = B.meth                # Pick B's method: classic
>>> x = D()
>>> x.meth()
A.meth
```

□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□□

□□□□□

Slot□□Python□□□

```
>>> class C:
...     __slots__ = ['a', 'b']                  # __slots__ means no __dict__ by default
...
>>> X = C()
>>> X.a = 1
>>> X.a
1
>>> X.__dict__
AttributeError: 'C' object has no attribute '__dict__'
>>> getattr(X, 'a')
1
>>> setattr(X, 'b', 2)                         # But getattr() and setattr() still work
>>> X.b
2
>>> 'a' in dir(X)                              # And dir() finds slot attributes too
True
>>> 'b' in dir(X)
True
```

□□□□□□□□□□□□□□□□□□□□□slots□□□□□□□□□□□□□□□□□

```
>>> class D:
...     __slots__ = ['a', 'b']
...     def __init__(self): self.d = 4          # Cannot add new names if no __dict__
...
>>> X = D()
AttributeError: 'D' object has no attribute 'd'
```

□□□□□□__slots__□□□__dict__□□□□□□□□□□□□□□□□□□□

```
>>> class D:
...     __slots__ = ['a', 'b', '__dict__']      # List __dict__ to include one too
...     c = 3                                   # Class attrs work normally
...     def __init__(self): self.d = 4          # d put in __dict__, a in __slots__
...
>>> X = D()
>>> X.d
4
>>> X.__dict__                                  # Some objects have both __dict__ and __slots__
{'d': 4}                                        # getattr() can fetch either type of attr
>>> X.__slots__
['a', 'b', '__dict__']
>>> X.c
3
>>> X.a                                         # All instance attrs undefined until assigned
AttributeError: a
>>> X.a = 1
>>> getattr(X, 'a',), getattr(X, 'c'), getattr(X, 'd')
(1, 3, 4)
```

□□□

```
>>> for attr in list(getattr(X, '__dict__', [])) + getattr(X, '__slots__', []):
...     print(attr, '=>', getattr(X, attr))

d => 4
a => 1
b => 2
__dict__ => {'d': 4}
```

□□□□□□__slot__□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__slots__□□□□slot□□□□□□□□□□□□□□□□□□□__slots__□□□□□□□□□□□□□□□□□□□

```
>>> class E:
...     __slots__ = ['c', 'd']                  # Superclass has slots
...
>>> class D(E):
...     __slots__ = ['a', '__dict__']           # So does its subclass
...
>>> X = D()
>>> X.a = 1; X.b = 2; X.c = 3                   # The instance is the union
>>> X.a, X.c
(1, 3)

>>> E.__slots__                                 # But slots are not concatenated
['c', 'd']
>>> D.__slots__
['a', '__dict__']
>>> X.__slots__                                 # Instance inherits *lowest* __slots__
['a', '__dict__']
>>> X.__dict__                                  # And has its own an attr dict
{'b': 2}

>>> for attr in list(getattr(X, '__dict__', [])) + getattr(X, '__slots__', []):
...     print(attr, '=>', getattr(X, attr))
...
b => 2                                          # Superclass slots missed!
a => 1
__dict__ => {'b': 2}

>>> dir(X)                                      # dir() includes all slot names
[...many names omitted... 'a', 'b', 'c', 'd']
```

□□□□□□□□□□□□□slots□□□□□□□□□□□□□

□□□

□□□□□□□□property□□□□□□□□

□□□

□□□

__getattribute__□□□□

__getattribute__□□□□

□□

□□□

□□□□□□□□□

□Python 2.2□□□□□□□□

□□□

Python 2.6□Python 3.0□□□□□□

□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□

·□Python 3.0□□□

```
C:\misc> c:\python26\python
>>> from spam import Spam
>>> a = Spam()                          # Cannot call unbound class methods in 2.6
>>> b = Spam()                          # Methods expect a self object by default
>>> c = Spam()

>>> Spam.printNumInstances()
TypeError: unbound method printNumInstances() must be called with Spam instance
as first argument (got nothing instead)
>>> a.printNumInstances()
TypeError: printNumInstances() takes no arguments (1 given)
```

□□□□□□□□□Python 2.6□□□□□□□□□

□□□□□□□Python 3.0□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□□□□□□□□□□self□□□Python 2.6□□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□□□self□□□□□□□□□

□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.2□□staticmethod□classmethod□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□

```
>>> obj = Methods()                    # Make an instance

>>> obj.imeth(1)                       # Normal method, call through instance
<__main__.Methods object...> 1         # Becomes imeth(obj, 1)

>>> Methods.imeth(obj, 2)              # Normal method, call through class
<__main__.Methods object...> 2         # Instance passed explicitly
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

```
class Sub(Spam):
    def printNumInstances():                    # Override a static method
        print("Extra stuff...")                 # But call back to original
        Spam.printNumInstances()
    printNumInstances = staticmethod(printNumInstances)
```

```
>>> a = Sub()
>>> b = Sub()
>>> a.printNumInstances()                       # Call from subclass instance
Extra stuff...
Number of instances: 2
>>> Sub.printNumInstances()                     # Call from subclass itself
Extra stuff...
Number of instances: 2
>>> Spam.printNumInstances()
Number of instances: 2
```

□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

```
class Spam:
    numInstances = 0                                # Trace class passed in
    def __init__(self):
        Spam.numInstances += 1
    def printNumInstances(cls):
        print("Number of instances:", cls.numInstances, cls)
    printNumInstances = classmethod(printNumInstances)

class Sub(Spam):
    def printNumInstances(cls):                     # Override a class method
        print("Extra stuff...", cls)                # But call back to original
        Spam.printNumInstances()
    printNumInstances = classmethod(printNumInstances)

class Other(Spam): pass                             # Inherit class method verbatim
```

□□□

□□□□□□□□□□□Other□□□□Spam□□□□□□□□□□□

```
class Spam:
    numInstances = 0
    def count(cls):                     # Per-class instance counters
        cls.numInstances += 1           # cls is lowest class above instance
    def __init__(self):
        self.count()                    # Passes self.__class__ to count
    count = classmethod(count)

class Sub(Spam):
    numInstances = 0
    def __init__(self):                 # Redefines __init__
        Spam.__init__(self)

class Other(Spam):                      # Inherits __init__
    numInstances = 0
>>> x = Spam()
>>> y1, y2 = Sub(), Sub()
>>> z1, z2, z3 = Other(), Other(), Other()
>>> x.numInstances, y1.numInstances, z1.numInstances
(1, 2, 3)
>>> Spam.numInstances, Sub.numInstances, Other.numInstances
(1, 2, 3)
```

□□□

□□□□□□□□□□□□

□□□□□□□□staticmethod□□□□□□□□□□□

```
class C:
   @staticmethod                                    # Decoration syntax
   def meth():
       ...
```

□□□

□□□staticmethod□□□

□□□□□

□□Python□□□□

□□spam□□□□□tracer□□□□□□□□□□□□□□□□□spam□□□□□□□□□□□□□□□□□□__call__□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□

```
def count(aClass):
    aClass.numInstances = 0
    return aClass                       # Return class itself, instead of a wrapper
@count
class Spam: ...                         # Same as Spam = count(Spam)

@count
class Sub(Spam): ...                    # numInstances = 0 not needed here

@count
class Other(Spam): ...
```

□□□

□□□□□□

□□□□□

□□□

□□□

```
>>> class X:
...     a = 1                                  # Class attribute
...
>>> I = X()
>>> I.a                                       # Inherited by instance
1
>>> X.a
1
```

□□□

□□□□□X□Y□□“□□□”□□□□□

□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□

```
x.shared.append('spam')          # Changes shared object attached to class in-place
x.shared = 'spam'                # Changed or creates instance attribute attached to x
```

□□□

□□□□□□□□□□

□□□□□□□□□

```
    def other(self): ...

class Super:
    def __str__(self): ...
    def other(self): ...

class Sub(ListTree, Super):              # Get ListTree's __str__ by listing it first
    other = Super.other                  # But explicitly pick Super's version of other
    def __init__(self):
        ...

x = Sub()                                # Inheritance searches Sub before ListTree/Super
```

□□□□□Sub□□□other□□□□□□□□□Sub.other——□Super.other□□□□□□□□□□□□□□□□□□□Sub.other□□□□□□ListTree.other□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Super□□□□□other□□□□□□□□□ListTree□□□□□

```
class Sub(Super, ListTree):              # Get Super's other by order
__str__ = Lister.__str__                 # Explicitly pick Lister.__str__
```

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□Python 2.2□□□□□□□□□□□

□□□□□□Python 2.2□□□□□□

```
def generate():
    return Spam()

class Spam:                                    # Define at top level of module
    count = 1
    def method(self):

        print(Spam.count)                      # Works: in global (enclosing module)

generate().method()
```

□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□

Python□□□□□□□□□__getattr__□□□□□

□□□□2 7□□□□□□□□3 0□□□□□□□□□□□□□□□□□□□□□□□□__getattr__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□

“□□□□”

□□□□□□□□□□OOP□□□□□□□□□□

□□□□

□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□

3.□□□□□□□□□□

4.□□□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□

□□□□ □□□

□□□

□□□

□□□

f.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__add__□□□□□□□□□□□□□□__getattr__□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□

```
class Lunch:
    def __init__(self)                    # Make/embed Customer and Employee

    def order(self, foodName)             # Start a Customer order simulation
    def result(self)                      # Ask the Customer what Food it has

class Customer:
    def __init__(self)                                  # Initialize my food to None
    def placeOrder(self, foodName, employee)            # Place order with an Employee
    def printFood(self)                                 # Print the name of my food

class Employee:
    def takeOrder(self, foodName)         # Return a Food, with requested name

class Food:
    def __init__(self, name)              # Store food name
```

□□□□□□□□□□

a.Lunch□□□□□□□□□□□□□□□□Customer□□□□□□Employee□□□□□□□□□□□□□order□□□□□□□□□□□□□□order□□□□□□Customer□□□□□□□□placeOrder□□□□□□Customer□place Order□□□□□□□□Employee□□□□□□Food□□□□□□□Empmloyee□takeOrder□□□

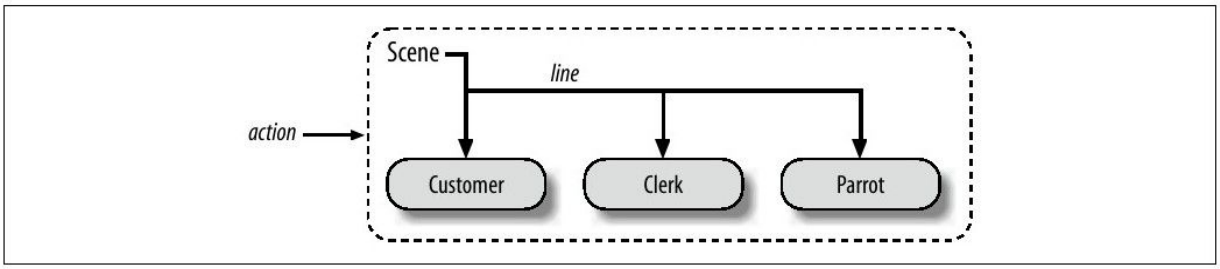
b.Food□□□□□□□□□□□□□□□□□□□□□□□"burritos"□□□Lunch.order□□□Customer.placeOrder□□□Employee.takeOrder□□□□Food□□□□□□□□Lunch□□□□□□□□□□result□□□□□□□□□□□□Employee□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□Lunch□□□□Employee□□□□□Customer□□□□Customer□□□Employee□□□□

□□□□□□□□□□□□□□□□□□□□□□□Lunch□□□□□□order□□□□□□□□□□□□□□□□result□□□□□Customer□□□□□□□□□□□□□□□□□□□

```
% python
>>> from zoo import Cat, Hacker
>>> spot = Cat()
>>> spot.reply()                    # Animal.reply; calls Cat.speak
meow
>>> data = Hacker()                 # Animal.reply; calls Primate.speak
>>> data.reply()
Hello world!
```

9.□□□□□□□□□□31-2□□□□□□□□



□ 31-2 □□□□□□□□□□□□□□□□□□□□□□□□Customer□Clerk□Parrot□□□□□□(Scene)□□□□□□□□□□□□□□□□□□□

```
% python
>>> import parrot
>>> parrot.Scene().action()                # Activate nested objects
customer: "that's one ex-bird!"
clerk: "no it isn't..."
parrot: None
```

□□□□□□□□□□□□□□OOP

□□□□□Python□□□□□□□□□□□□□

□□

□□□

□□□□□□□□□□□□OOP□□□□□□□□□□□□

□□□□ □□□□□

□32□ □□□□

□□Python□□□□□□□□□□□□□□□□□□□

□Python 2.6□□□□□□□□□□□□□□□□□□□□2.5□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□

□Python□□□□

□□□□□□□□□□□□□□□"goto"□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>> fetcher(x, 4)                                   # Default handler - shell interface
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in fetcher
IndexError: string index out of range
```

□□□

□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□Python□□□□□□□□□□

```
>>> assert False, 'Nobody expects the Spanish Inquisition!'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>


AssertionError: Nobody expects the Spanish Inquisition!
```

□□□□□□□

□□□□□□□□raise□□□□Python□□□□□□□□

□□□□

□□□try□□□□□"finally"□□□□□□□□□□□finally□□□□□□□□□□□□□□□□except□□□□□□try/finally□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□

```
>>> try:
...     fetcher(x, 3)
... finally:                                    # Termination actions
...     print('after fetch')
...
'm'
after fetch
>>>
```

□□□□□□try□□□□□□□□□□□finally□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

```
>>> def after():
...     try:
...         fetcher(x, 4)
...     finally:
...         print('after fetch')
...     print('after try?')
...
>>> after()
after fetch
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in after
  File "<stdin>", line 2, in fetcher
IndexError: string index out of range
>>>
```

□□□□□□□□□□"after try?"□□□□□□□□□□□□□□□□□try/finally□□□□□□□□□□□□□□□Python□□□□□finally□□□□□□□□□□□□□□□□□

```
>>> def after():
...     try:
...         fetcher(x, 3)
...     finally:
...         print('after fetch')
...     print('after try?')
...
>>> after()
after fetch
after try?
>>>
```

□□□□□□□try/except□□□□□□□□□□□□□□□□□□□try/finally□□□□□□
□□□□□□□□try□□
try/except□□□□□□□□□□□□□□□□□□□□□□□□□□□try/finally□□□□□□□□□□□□□
□□

□□□□□□□□□□□□□□□□□□□□□□□□Python 2.5□□□□□□□□□□□try□□□□□
except□finally□□□finally□□□□□□□□□□□□□□□□□□□□□□□□□□□□□except
□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□Python 3.0□□□
try/finally□□□□□□with/as□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> with open('lumberjack.txt', 'w') as file:                # Always close file on exit
...     file.write('The larch!\n')
```

□□□

```
def doStuff():                          # Python code
    doFirstThing()                      # We don't care about exceptions here,
    doNextThing()                       # so we don't need to detect them
    ...
    doLastThing()

if __name__ == '__main__':
    try:
        doStuff()                       # This is where we care about results,
    except:                             # so it's the only place we must check
        badEnding()
    else:
        goodEnding()
```

□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□3□□□□

2.□□□□□□□□

□□□□

1.□□□

第33章　异常处理

异常处理是现代程序设计语言必须具备的特性之一——本章将Python异常处理的相关知识，包括try、raise、assert、with等关键字的使用，以及Python内建异常类的层次结构。

本章将介绍Python 2.5中finally关键字与try关键字和except、else关键字联合使用的方法，Python 3.0和Python 2.6中引入的with语句的使用方法，以及Python 3.0中raise语句和except语句的变化，并对Python 2.6和Python 3.0中异常处理的实现进行简单的介绍。

try/except/else语句

本节将介绍Python异常处理中try/except/else和try/finally语句的使用方法，以及Python 2.5中这两个语句的变化。Python 2.5中except和finally关键字与try关键字的联合使用。

try□□□□□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□except□□□□□□□□□□□□□□□□□□□□□□□□□□□else□□□try□except□□else□□□□□□□□□□□□□□□□□□□

·□□□□□□□try□□□□□□□□□except□□□□□□□□□

表33-1：try语句分句形式

分句形式	说明
except:	捕捉所有（其他）异常类型
except *name*:	只捕捉特定的异常
except *name*, *value*:	捕捉所列的异常和其额外的数据（或实例）
except (*name1*, *name2*):	捕捉任何列出的异常
except (*name1*, *name2*), *value*:	捕捉任何列出的异常，并取得其额外数据
else:	如果没有引发异常，就运行
finally:	总是会运行此代码块

□□□□□raise□□□□□

```
try:
    action()
except NameError:
    ...
except IndexError:
    ...
except KeyError:
    ...
except (AttributeError, TypeError, SyntaxError):
    ...
else:
    ...
```

□□□□□□□□□action□□□□□□□□□□□□□Python□□□try□□□□□□□□□□□□□except□Python□□□□□□□□□□□□□except□□□□□□□□□□□□□except□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□action□□□□□□□□□□else□□□□□□□□□except□□□□□□□□□□□□□

□□□□□□□□□“□□□□”□□□□□except□□□□□□

```
try:
   action()
except NameError:
    ...                                    # Handle NameError
except IndexError:
    ...                                    # Handle IndexError
except:
    ...                                    # Handle all other exceptions
else:
    ...                                    # Handle the no-exception case
```

□□except□□□

3.0□□□as□□□

try/else□□

Python□□□□□□□□else□□□□□□□□□□□□□else□□□□□□□□□□□□□

```
try:
    ...run code...
    ...no exception occurred...
except IndexError:
    ...handle exception...
```

□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□”□□□□□□□□Index-Error□□□□□try□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□else□□□□□□□□□□□□□□□□except□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□□else□□□□□□□□□□□□□□

□□□□□□□

□□Python□□□□□□

□□Python 3.0□□□□shell□□□□

□□□□□□□□□

Python□□□□□□□□

□□□

try/finally□□

try□□□

□□□□□Python 2.4□□□□□□□□finally□□□□□except□else□□□□□□□□try□□□□□□□□□□□□□□□□□□□□try/finally□□□□□□□□□□□□□□□□□□□□Python 2.5□finally□□□except□else□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

□□□□□□□9□□□□□□□□□□□□□□□□□□

□□try/except/finally□□

□Python 2.5□□□□□□□□□□□□□□□□□□□□□15□□□□□□□□□□□Python□□□□try□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□finally□□□□□□□□□□□□□□□□□□□except□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□else□□□□□□□□□□□□

□□□□□finally□□□□□except□else□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□main-action□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□except□□□□□□□□□□□□□□□□□□□

□□try□□□□

□□□□□□□□□□try□□□□□□□except□□□finally□□□□□□□□□□□□□□□□□□□

```
try -> except -> else -> finally
```

□□□else□finally□□□□□□□□□0□□□□except□□□□□□□□□□else□□□□□□□□□except□□□□□□try□□□□□□□□□□□□□□□□else□except□□□□□finally□

□□□□□□□□□□□□□□□□□□□

□□□except□□□□□□□□□□□□□□□□□□□□□□□□□raise□□□□□□□□□□□□□□□□□□□

□□□□□□finally□except

□Python 2.5□□□□□□□□try□□□finally□except□□□□□□□□□□□□□ try/finally□□□try□□□□□□try/except□□35□□□□□□□□□□□□□

□□try□□□

□□□□□□□□□try□□□□□□□□□□□□□□□□mergedexc.py□□□□□□□□□□□□□□□□

```
sep = '-' * 32 + '\n'
print(sep + 'EXCEPTION RAISED AND CAUGHT')
try:
    x = 'spam'[99]
except IndexError:
    print('except run')
finally:
    print('finally run')
print('after run')

print(sep + 'NO EXCEPTION RAISED')
try:
    x = 'spam'[3]
except IndexError:
    print('except run')
finally:
    print('finally run')
print('after run')

print(sep + 'NO EXCEPTION RAISED, WITH ELSE')
try:
    x = 'spam'[3]
except IndexError:
    print('except run')
else:
    print('else run')
```

```
finally:
    print('finally run')
print('after run')

print(sep + 'EXCEPTION RAISED BUT NOT CAUGHT')
try:
    x = 1 / 0
except IndexError:
    print('except run')
finally:
    print('finally run')
print('after run')
```

□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□

```
c:\misc> C:\Python30\python mergedexc.py
--------------------------------
EXCEPTION RAISED AND CAUGHT
except run
finally run
after run
--------------------------------
NO EXCEPTION RAISED
finally run
after run
--------------------------------
NO EXCEPTION RAISED, WITH ELSE
else run
finally run
after run
--------------------------------
EXCEPTION RAISED BUT NOT CAUGHT
finally run
Traceback (most recent call last):
  File "mergedexc.py", line 36, in <module>
    x = 1 / 0
ZeroDivisionError: int division or modulo by zero
```

□□□□□□main-action□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

raise□□

□□□□□□□□□□□□□□raise□□□□□□□□□□□□□□raise□□□□□□□raise□□□□□□□□□□□□□□

```
exc = IndexError()                          # Create instance ahead of time
raise exc

excs = [IndexError, TypeError]
raise excs[0]
```

□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□except name as X:□□□□□X□□□□□□□□□□□□□□□□□

```
try:
    ...
except IndexError as X:                     # X assigned the raised instance object
    ...
```

as□try□□□□□□□□□□□□□□□□□□□

□□□except□□□□□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□raise□□□□□□□□□□□

□□raise□□□□

raise□□□□□□□□□□□□□□□□□□□

Python 3.0□□□□raise from

Python 3.0□□□□Python 2.6□□□□raise□□□□□□□□□□from□□□

```
raise exception from otherexception
```

□□□from□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__cause__□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> try:
...     1 / 0
... except Exception as E:
...     raise TypeError('Bad!') from E
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ZeroDivisionError: int division or modulo by zero
```

□□□□□□□□□□□□□□□□□

```
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>

TypeError: Bad!
```

□□□

assert□□

Python□□□□assert□□□□□□□□□□□□□□□raise□□□□□□□□□□□□□□ assert□□□□□□□raise□□□□□□□□□□

```
assert <test>, <data>                    # The <data> part is optional
```

□□□□□□□□□□□□□

```
if __debug__:
    if not <test>:
        raise AssertionError(<data>)
```

□□□□□□□test□□□□□Python□□□□□□□data□□□□□□□□□□□□□□□□□□□□□□□□□□□□□AssertionError□□□□□□try□□□□□□□□□□□□□□□□□□□

Assert□□□

[1]□□□

with/as□□□□□

Python 2.6□Python 3.0□□□□□□□□□□□□□□□□□with□□□□□as□□□□□□□□□□□□□□□□□

□□□□expression□□□□

```
myfile = open(r'C:\misc\data')
try:
    for line in myfile:
        print(line)
        ...more code here...
finally:
    myfile.close()
```

□□□□□□□□□Python□□□□□□□□□

□□□

□□□□□□

□□□□□

□□□

```
% python withas.py
starting with block
running test 1
reached
exited normally

starting with block
running test 2
raise an exception! <class 'TypeError'>
Traceback (most recent call last):
  File "withas.py", line 20, in <module>
    raise TypeError
TypeError
```

□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□

□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□try□□□□raise□□□□assert□□□□□□□□□with□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□

1.try□□□□□□□□

2.try□□□□□□□□□□□□□□

3.raise□□□□□□□□

4.assert□□□□□□□□□□□□□□□□□□□□□□

5.with/as□□□□□□□□□□□□□□□□□□□□□

□□□□

1.try□□□

□34□　□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□Exception□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

```
C:\misc> C:\Python25\python
>>> myexc = "My exception string"                # Were we ever this young?
>>> try:
...     raise myexc
... except myexc:
...     print('caught')
...
caught
```

□□□

□□□□□□

□□□□□

□□□

□□□□□

□□□□

Exception□□

□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□sys.exc_info□□□□□□□□□□□□□□□□except□□□except□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□except□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

```
# client.py

try:
    mathlib.func(...)
except (mathlib.Divzero, mathlib.Oflow, mathlib.Uflow):
    ...handle and recover...
```

□□□

□□□

```
# mathlib.py

...
class Uflow(NumErr): pass
```

□□□

□□Exception□

□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

GeneratorExit□□□

□□□

□□□□□□□

□□□□□□

```
>>> class E(Exception): pass
...
>>> try:
...     raise E('spam')
... except E as X:
...     print(X, X.args)                          # Displays and saves constructor arguments
...
spam ('spam',)

>>> try:
...     raise E('spam', 'eggs', 'ham')
... except E as X:
...     print(X, X.args)
...
('spam', 'eggs', 'ham') ('spam', 'eggs', 'ham')
```

□□□

□□□□□□

□□□□□

```
>>> class MyBad(Exception):
...     def __str__(self):
...         return 'Always look on the bright side of life...'
...
>>> try:
...     raise MyBad()
... except MyBad as X:
...     print(X)
...
Always look on the bright side of life...

>>> raise MyBad()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.MyBad: Always look on the bright side of life...
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__str__□□□□□□□□□□□□□□□
__str__□□□□□□□□□□□□□□□□□□□□□__str__□□□__repr__□□□□□□□□□□
__repr__□□□□□□□□□□□□□□□__str__□□□□29□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

```
>>> class FormatError(Exception):
...     def __init__(self, line, file):
...         self.line = line
...         self.file = file
...
>>> def parser():
...     raise FormatError(42, file='spam.txt')                    # When error found
...
>>> try:
...     parser()
... except FormatError as X:
...     print('Error at', X.file, X.line)
...

Error at spam.txt 42
```

□□□□except□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□X□□□[1]□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□

```
C:\misc> C:\Python30\python parse.py
C:\misc> type formaterror.txt
Error at spam.txt 40
```

□□□□□□□□□□□□□□logerror□□□□□□□□□□□□□□□□□□□□□□line□file□□□□□□□□□□□□□□□□□□□

□□□□

□□□

□□□□

1.□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□

3.□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.□□□□□□□□□□□□□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□Python 3.0□□□

□35□　□□□□□

□□□□□

□□□

□ 35-2 □□□try/finally□□□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□finally□□□□□□□□□□□□□□□□□□□□try□□□finally□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Finally□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□action1□□□□action2□□TypeError□□□□□□□□□□□□□try□□□□□□□action1□□□□□□□□□□□□□□□□Python□□□□□□□□□□□except□□□□try□□□□□□□□□□□□action1□□try□

□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□33□□□□□□□□□□try/except/finally□□□□□□□□□□

```
>>> try:
...     try:
...         raise IndexError
...     finally:
...         print('spam')
... finally:
...     print('SPAM')
...
spam
SPAM
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
IndexError
```

□□□35-2□□except-finally.py□

```
def raise1(): raise IndexError
def noraise(): return
def raise2(): raise SyntaxError

for func in (raise1, noraise, raise2):
    print('\n', func, sep='')
    try:
        try:
            func()
        except IndexError:
            print('caught IndexError')
    finally:
       print('finally run')
```

□□finally□□□□□□□□□□□□□□□□□□□

```
% python except-finally.py
<function raise1 at 0x026ECA98>
caught IndexError
finally run

<function noraise at 0x026ECA50>
finally run

<function raise2 at 0x026ECBB8>
finally run

Traceback (most recent call last):
  File "except-finally.py", line 9, in <module>
    func()


  File "except-finally.py", line 3, in raise2
    def raise2(): raise SyntaxError
SyntaxError: None
```

□□□□□□33□□□□□□□Python 2.5□□except□finally□□□□□□□□□try□□□□□□□□□

□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□sys.exit()□□□□□□□Ctrl-C□□□□□□SystemExit□KeyboardInterrupt□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□raise

□□□□□□

```
class Failure(Exception): pass

def searcher():
    if ...success...:
        return ...founditem...
    else:
        raise Failure()

try:
    item = searcher()
except Failure:
    ...report...
else:
    ...use item here...
```

□□Python□□□

□□□□□□□□□□

□□□□33□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□try/finally□□□□□□□□□□□□□□□□□□□

□try□□□□□

□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□doctest□
PyUnit□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□sys.exc_info

sys.exc_info□□except□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
try:
    ...
except:
    # sys.exc_info()[0:2] are the exception class and instance
```

□□□□□□□□□□□□□□□□□□□□□□□□None□□□□□□□□□□□□□□□type□value□traceback□□

·type□□□□□□□□□□□□□□□

·value□□□□□□□□□□□

·traceback□□□traceback□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□traceback□□□□□□□□□□□□□□□□□□□

```
try:
    ...
except General as instance:
    # instance.__class__ is the exception class
```

□□□

□□□□□□□□□

□□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

□□□□□□□□except□□

□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□except□□□□□try□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```
import sys
def bye():
    sys.exit(40)                          # Crucial error: abort now!
try:
    bye()
except:
    print('got it')                       # Oops--we ignored the exit
print('continuing...')

% python exiter.py
got it
continuing...
```

□□□

```
mydictionary = {...}
...
try:
    x = myditctionary['spam']                    # Oops: misspelled
except:
    x = None                                     # Assume we got KeyError
...continue here with x...
```

□□□

□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□

□□□□□□□Python□□□□□□□

□□□□□□□□□□□□□□□□□□Python□□□□□C□C++□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□Python□□□□□□□□□□Python□□□□□□□□□□

PyChecker□PyLint□□□□□□□□□□□□□□□□http://www.python.org□□
PyPI□□□□□□□□□□□□□□□□□Web□□□□□□□□□

PyUnit□□□□unittest□

□□24□□□□□□□□□□□□□□□□□□__name__=='__main__'□□□□□□□□
□□□□Python□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□
PyUnit□□□□□□□□unittest□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□Java□JUnit□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□

doctest

doctest□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□
□□□□□□□□□□□□□□□□□doctest□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□doctest□□□
□□□□doctest□□□□□□□□□□□□□□□□□□□□□□□□□□□□

IDE

□□□□3□□□□Python□IDE□□□IDLE□□IDE□□□□□□□□□□□□□□□□□□□□
□□□□□Python□□□□□□□□IDE□□□□Eclipse□Komodo□NetBeans□
Wing IDE□□□□□□□□□□□□□□□□□□□□□□□□□□□□GUI□□□□□□□□□□□□□□□□□3□□
http://www.python.org□text editor□□□□□□□□□Web□□□□□□□□□□
Python□□□IDE□GUI□□□□□

□□□□

□□Python□□□□

□□□□□□□□□pdb□□□□□□□□□□□□□C□□□□□□□□□□dbx□gdb□□□□□□□□□□□□□□□□

□□□□□□□□pdb□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□2□□□□□□Psyco□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□Shedskin□□□Python□C++□□□□□□□□□□□.pyo□□□□□□□□□□□□□□-O Python□□□□□□□□□□□□□□□□□□21□□□33□□□□□□□□□□□□□□□□□□

□□□□

□□□

□□□□ □□□

□□□

4.□□□□□□□□□B□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□ □□□□

□36□ Unicode□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□7□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□Python□□□□□□□□□□□□□ASCII□□□□□□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□struct□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□Python 3.0 □□□□□□□□□□□□□□□□□□□□

□□□

□□□

□□□□□□□□□□□□□□□latin-1□iso_8859_1□8859□□□□□□□□□□□□Latin-1□□□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□

□□□□□Unicode□□□□□□□□□Python□□□□□□□□"Python HOWTOs"□□□□□□□□"Unicode HOWTO"□□□□□□□□□□□□□□□□□□□

Python□□□□□□

□□□□□Python□□□□□□

Python 3.0□□□3□□□

bytes□□□□□□□□list□□□□□□□□

□□□□□□□□□

□□I/O□□□□□□□□□Python 3.0□□□□□□□□□□□str/bytes□□□□□□□□□Unicode□□□□□□□□Python□□

□□□□

□□□□□□□□□□□□□□□□□□□

·□□□

□□□□□□□□□□□“□Python 3.0□□□BOM”□□□□□□□□□□BOM□□□□□□□□□□□□□□□□Python□□□Unicode□□□□□□□□□□

Python 3.0□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□str□□□□□□□□□ASCII□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□□□□□□7□□□□□□□□□□□□□□□□□□Python 2.6□□□bytes□□□□□□□□□□str□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□bytes(X)□□str(X)□□□□□□□□□□□□□□□□b'...'□□□□□□'...'□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python

>>> B = b'spam'                  # Make a bytes object (8-bit bytes)
>>> S = 'eggs'                   # Make a str object (Unicode characters, 8-bit or wider)

>>> type(B), type(S)

(<class 'bytes'>, <class 'str'>)

>>> B                           # Prints as a character string, really sequence of ints
b'spam'
>>> S
'eggs'
```

bytes□□□

```
>>> B[0] = 'x'                                    # Both are immutable
TypeError: 'bytes' object does not support item assignment

>>> S[0] = 'x'
TypeError: 'str' object does not support item assignment

>>> B = B"""          # bytes prefix works on single, double, triple quotes
... xxxx
... yyyy
... """
>>> B
b'\nxxxx\nyyyy\n'
```

□□□□□□□□□Python 2.6□□□□□□□□□□b 'xxx'□□□□□'xxx'□□□□□□□□□□str□□□□bytes□□str□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□bytes□□□□□□□□□□□□□□□Python 2.6□□u 'xxx'□U 'xxx' Unicode□□□□□□□□Python 3.0□□□□□□□□□□□□'xxx'□□□□□□□□□□□□□□□Unicode□□□□□□□□□□ASCII□□□□□□□□□“□□□ASCII□□”□□□□□□□□□□□

□□

□□Python 2.X□□str□unicode□□□

```
>>> S = 'eggs'
>>> S.encode()                          # str to bytes: encode text into raw bytes
b'eggs'

>>> bytes(S, encoding='ascii')          # str to bytes, alternative
b'eggs'

>>> B = b'spam'
>>> B.decode()                          # bytes to str: decode raw bytes into text
'spam'

>>> str(B, encoding='ascii')            # bytes to str, alternative
'spam'
```

□□□□□□□□□□□□□□□□□□□□□□□sys□□□□□□□□□bytes□□□□□□□□□□□□□□□□□□□str.encode(□bytes.decode)□□□□□□

□□□□□□□str□□□bytes□□□□□□□□□□□□str□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□str□□□□bytes□□□□□□□□□□□□□□□□str□□□□□□□□□□□□□□□□□□□□□□□□□□□□□B□S□□□□□□□:

```
>>> import sys
>>> sys.platform                        # Underlying platform
'win32'
>>> sys.getdefaultencoding()            # Default encoding for str here
'utf-8'

>>> bytes(S)
TypeError: string argument without an encoding

>>> str(B)                              # str without encoding
"b'spam'"                               # A print string, not conversion!
>>> len(str(B))
7
>>> len(str(B, encoding='ascii'))       # Use encoding to convert to str
4
```

□□Unicode□□□

□□□□□□□□□□□□ASCII Unicode□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□"\xNN"□□□□□□□□□□□□"\uNNNN"□"\UNNNNNNNN " Unicode□□□□Unicode□□□□□□□□□□□□□□4□□□□□□□□□□1□2□□□16□□□□□□□□□□□□□□□□8□□□□□□□□□4□□□32□□□□□□

□□ASCII□□

□□□□□□□□□□□□□□□□□□□

□□□□□□7□ASCII□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> S.encode('ascii')          # Values 0..127 in 1 byte (7 bits) each
b'XYZ'
>>> S.encode('latin-1')        # Values 0..255 in 1 byte (8 bits) each
b'XYZ'
>>> S.encode('utf-8')          # Values 0..127 in 1 byte, 128..2047 in 2, others 3 or 4
b'XYZ'
```

□□□□□□□□□□□ASCII□□□□□□bytes□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ASCII□□□

```
>>> S.encode('latin-1')[0]
88
>>> list(S.encode('latin-1'))
[88, 89, 90]
```

□□□ASCII□□

□□□□ASCII□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□

□□□□□□ASCII□□

□□□□□□□□□□□□□□ASCII□□□□□□raw□□□□ASCII□□□□□□□□□□□□□□□□□□□Latin-1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□UTF-8□□□□□□□□□□□□□□□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□raw□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> S = '\u00c4\u00e8'
>>> S
'Äè'
>>> len(S)
2

>>> S.encode('ascii')
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1:
ordinal not in range(128)

>>> S.encode('latin-1')                    # One byte per character
b'\xc4\xe8'

>>> S.encode('utf-8')                      # Two bytes per character
b'\xc3\x84\xc3\xa8'

>>> len(S.encode('latin-1'))               # 2 bytes in latin-1, 4 in utf-8
2
>>> len(S.encode('utf-8'))
4
```

□□□□□□□□□□□□□□□□□□□raw□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□open□□□□□□□□□□□□□□□□□□□

```
>>> B = b'\xc4\xe8'
>>> B
b'\xc4\xe8'

>>> len(B)                                  # 2 raw bytes, 2 characters
2
>>> B.decode('latin-1')                     # Decode to latin-1 text
'Äè'

>>> B = b'\xc3\x84\xc3\xa8'
>>> len(B)                                  # 4 raw bytes
4
>>> B.decode('utf-8')
'Äè'
>>> len(B.decode('utf-8'))                  # 2 Unicode characters
2
```

□□Unicode□□□□

□□□□

```
b'\xc1c\xc2T\xc3'
>>> S.encode('cp850')                    # 5 bytes each
b'A\x8eB\x8aC'

>>> S = 'spam'                           # ASCII text is the same in most
>>> S.encode('latin-1')
b'spam'
>>> S.encode('utf-8')
b'spam'
>>> S.encode('cp500')                    # But not in cp500: IBM EBCDIC!
b'\xa2\x97\x81\x94'
>>> S.encode('cp850')
b'spam'
```

□□□□□□□□□□□□□chr□□□Unicode□□□□□□□□□□□□Unicode□□

```
>>> S = 'A' + chr(0xC4) + 'B' + chr(0xE8) + 'C'
>>> S
'AÄBèC'
```

□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□Unicode□□□□□□□□str□□□□□□□□□□□□□□□□□□□□□□□□□□□bytes□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bytes□□□□□str□□□□□□□□□□□□□ASCII□□□

```
>>> S = 'A\xC4B\xE8C'                    # str recognizes hex and Unicode escapes
>>> S
'AÄBèC'

>>> S = 'A\u00C4B\U000000E8C'
>>> S
'AÄBèC'

>>> B = b'A\xC4B\xE8C'                   # bytes recognizes hex but not Unicode
>>> B
b'A\xc4B\xe8C'

>>> B = b'A\u00C4B\U000000E8C'           # Escape sequences taken literally!
>>> B
b'A\\u00C4B\\U000000E8C'

>>> B = b'A\xC4B\xE8C'                   # Use hex escapes for bytes
>>> B                                    # Prints non-ASCII as hex
b'A\xc4B\xe8C'
>>> print(B)
b'A\xc4B\xe8C'
>>> B.decode('latin-1')                  # Decode as latin-1 to interpret as text
'AÄBèC'
```

□□□□□□□□□□□□□□□ASCII□□□□□□□□□□□□□□127□□□□□□□□□□□□str□□□□□□□□□□□□□□□

```
>>> S = 'AÄBèC'                          # Chars from UTF-8 if no encoding declaration
>>> S
'AÄBèC'

>>> B = b'AÄBèC'
SyntaxError: bytes can only contain ASCII literal characters.

>>> B = b'A\xC4B\xE8C'                   # Chars must be ASCII, or escapes
>>> B
b'A\xc4B\xe8C'
>>> B.decode('latin-1')
'AÄBèC'

>>> S.encode()                           # Source code encoded per UTF-8 by default
b'A\xc3\x84B\xc3\xa8C'                   # Uses system default to encode, unless passed
>>> S.encode('utf-8')
b'A\xc3\x84B\xc3\xa8C'

>>> B.decode()                           # Raw bytes do not correspond to utf-8
UnicodeDecodeError: 'utf8' codec can't decode bytes in position 1-2: ...
```

□□□□

□□□

□Python 2.6□□□Unicode□□□

□□□□□□□Python 3.0□□□□Unicode□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□unicode□Python 2.6□□□□□□□□□□□str□□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□□□□□□raw□□□□□□□□Unicode□□□□□□□□□□□□□□□□□Python 2.6□str□□Python 3.0□bytes□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□

```
C:\misc> c:\python26\python
>>> import sys
>>> sys.version
'2.6 (r26:66721, Oct 2 2008, 11:35:03) [MSC v.1500 32 bit (Intel)]'

>>> S = 'A\xC4B\xE8C'                    # String of 8-bit bytes
>>> print S                              # Some are non-ASCII
AÄBèC

>>> S.decode('latin-1')                  # Decode byte to latin-1 Unicode
u'A\xc4B\xe8C'

>>> S.decode('utf-8')                    # Not formatted as utf-8
UnicodeDecodeError: 'utf8' codec can't decode bytes in position 1-2: invalid data

>>> S.decode('ascii')                    # Outside ASCII range
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc4 in position 1: ordinal
not in range(128)
```

□□□□□□□□□□Unicode□□□□u 'xxx'□□□□□□□□unicode□□□□□□□□Python 3.0□□□□□□□□□Python 3.0□□□□□□□□□Unicode□□

```
>>> U = u'A\xC4B\xE8C'                    # Make Unicode string, hex escapes
>>> U
u'A\xc4B\xe8C'
>>> print U
AÄBèC
```

□□□□□□□□□□Unicode□□□□□□□□raw□□□□□□□□□□Python 3.0□□str□□□□□bytes□□□

```
>>> U.encode('latin-1')                   # Encode per latin-1: 8-bit bytes
'A\xc4B\xe8C'
>>> U.encode('utf-8')                     # Encode per utf-8: multibyte
'A\xc3\x84B\xc3\xa8C'
```

□Python 2.6□□□ASCII□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□Python 3.0□□bytes□□□□Python 2.6□□"\u..."□"\U..."□□□□□□□unicode□□□□□□□8□str□□□□

```
C:\misc> c:\python26\python
>>> U = u'A\xC4B\xE8C'                          # Hex escapes for non-ASCII
>>> U
u'A\xc4B\xe8C'
>>> print U
AÄBèC

>>> U = u'A\u00C4B\U000000E8C'                  # Unicode escapes for non-ASCII
>>> U                                           # u'' = 16 bits, U'' = 32 bits
u'A\xc4B\xe8C'
>>> print U
AÄBèC

>>> S = 'A\xC4B\xE8C'                           # Hex escapes work
>>> S
'A\xc4B\xe8C'
>>> print S                                     # But some print oddly, unless decoded
A-BFC
>>> print S.decode('latin-1')
AÄBèC

>>> S = 'A\u00C4B\U000000E8C'                   # Not Unicode escapes: taken literally!
>>> S
'A\\u00C4B\\U000000E8C'
>>> print S
A\u00C4B\U000000E8C
>>> len(S)
19
```

□□Python 3.0□□str□bytes□□□Python 2.6□unicode□str□□□

□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□re□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

Unicode□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ASCII□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□UTF-8□□□□□□□□□□□□□□□□□□□

```
# Any of the following string literal forms work in latin-1.
# Changing the encoding above to either ascii or utf-8 fails,
# because the 0xc4 and 0xe8 in myStr1 are not valid in either.

myStr1 = 'aÄBèC'

myStr2 = 'A\u00c4B\U000000e8C'

myStr3 = 'A' + chr(0xC4) + 'B' + chr(0xE8) + 'C'

import sys
print('Default encoding:', sys.getdefaultencoding())

for aStr in myStr1, myStr2, myStr3:
print('{0}, strlen={1}, '.format(aStr, len(aStr)), end='')

bytes1 = aStr.encode()                    # Per default utf-8: 2 bytes for non-ASCII
bytes2 = aStr.encode('latin-1')           # One byte per char
#bytes3 = aStr.encode('ascii')            # ASCII fails: outside 0..127 range

print('byteslen1={0}, byteslen2={1}'.format(len(bytes1), len(bytes2)))
```

□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python text.py
Default encoding: utf-8
aÄBèC, strlen=5, byteslen1=7, byteslen2=5
AÄBèC, strlen=5, byteslen1=7, byteslen2=5
AÄBèC, strlen=5, byteslen1=7, byteslen2=5
```

□□□□□□□□□□□□□□□□□□UTF-8□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□

□□Python 3.0 Bytes□□

□□□□7□□□□□□□□□Python 3.0□□str□□□□□□□□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python

# Attributes unique to str

>>> set(dir('abc')) - set(dir(b'abc'))
{'isprintable', 'format', '__mod__', 'encode', 'isidentifier',
'_formatter_field_name_split', 'isnumeric', '__rmod__', 'isdecimal',
'_formatter_parser', 'maketrans'}

# Attributes unique to bytes

>>> set(dir(b'abc')) - set(dir('abc'))
{'decode', 'fromhex'}
```

□□□□□□□□str□bytes□□□□□□□□

□□□□□□□□□□□□□□□□□□□□Python 3.0□□str□□□□bytes□□□□□□□□□□□7□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> b'%s' % 99
TypeError: unsupported operand type(s) for %: 'bytes' and 'int'

>>> '%s' % 99


'99'

>>> b'{0}'.format(99)
AttributeError: 'bytes' object has no attribute 'format'

>>> '{0}'.format(99)
'99'
```

□□□□

□□□□□□□□□□□□□□Python 2.X□□□□□□□□□□□□□□

□□bytes□□□□□□□

□□□□□□□□□□□□□□□□b'...'□□□□□□bytes□□□□□□□□□□□□□□□□□str□□□□□□□□□bytes□□□□□□□□□□□□□□□□□□□□□□□□□□□□bytes□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□□

□□□□“□□□□”□□□replace□□□□□□□□□□□□□bytes□□□str□□□□□□□□□□□Python 2.X□□□□□□str□unicode□□□□□□□□□str□□□7□ASCII□□□□□□Python 3.0□□□□□□□□□□□□□

```
# Must convert manually in mixed-type expressions

>>> b'ab' + 'cd'
TypeError: can't concat bytes to str

>>> b'ab'.decode() + 'cd'                          # bytes to str
'abcd'

>>> b'ab' + 'cd'.encode()                          # str to bytes
b'abcd'

>>> b'ab' + bytes('cd', 'ascii')                   # str to bytes
b'abcd'
```

□□□□□□□□□bytes□□□□□□□□□

□□Python 3.0□□Python 2.6□bytearray□□

□□□□□□□□□□□□□□□str□bytes□□□□□□□□Python 2.X□unicode□str□□□□Python 3.0□□□□□□□□□□□bytearray□□□□□□0□255□□□□□□□□□□□□□□□□□□□bytes□□□□□□□□□□□□□□□□bytes□□□□□□□□□□□□□□□□□□□

```
# Creation in 3.0: text/binary do not mix

>>> S = 'spam'
>>> C = bytearray(S)
TypeError: string argument without an encoding

>>> C = bytearray(S, 'latin1')          # A content-specific type in 3.0
>>> C


bytearray(b'spam')

>>> B = b'spam'                         # b'..' != '..' in 3.0 (bytes/str)
>>> C = bytearray(B)
>>> C
bytearray(b'spam')
```

□□□□□bytearray□□□bytes□□□□□

□□bytearray□□□

□□□□□□□□□□□□□□□□□□□bytearrays□□□□□□□□□□□□□□□□□□□

```
# Binary versus text

>>> B                                                    # B is same as S in 2.6
b'spam'
>>> list(B)
[115, 112, 97, 109]
```

```
>>> C
bytearray(b'xYamLMNO')
>>> list(C)
[120, 89, 97, 109, 76, 77, 78, 79]

>>> S
'spam'
>>> list(S)
['s', 'p', 'a', 'm']
```

□□Python 3.0□□□

□□□□□□□□□□□□□

□□□□□□□Python 3.0□□□□□□□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□I/O□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ASCII□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□Python 2.X□□□□□□□□□□□□□□□□□□□

Python 3.0□□□□□□□□□□

□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□str□□□□□□□□□□□□□□□□□Windows□□□□□□□□□\n□□□□□□\r\n□□□□□□□□□□□□□□□□□□□

□□□□Windows□□□□□□□□□□□□□□□□□□□□\n□□□□□□□□\r\n□□□□□□□□□□□□\r\n□□□\n□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□bytes□□□□□□□□□□□□□□bytes□bytearray□□□□□□□□□□bytearray□□□□bytes□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□API□□□□bytes□□□□□□bytearray□

```
# bytearrays work too

>>> BA = bytearray(b'\x01\x02\x03')

>>> open('temp', 'wb').write(BA)
3

>>> open('temp', 'r').read()
'\x01\x02\x03'

>>> open('temp', 'rb').read()
b'\x01\x02\x03'
```

□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□Python□str/bytes□□□□□□□□□□□□□□□□□□□

```
# Can't read truly binary data in text mode

>>> chr(0xFF)                                   # FF is a valid char, FE is not
'ÿ'


>>> chr(0xFE)
UnicodeEncodeError: 'charmap' codec can't encode character '\xfe' in position 1...

>>> open('temp', 'w').write(b'\xFF\xFE\xFD')     # Can't use arbitrary bytes!
TypeError: can't write bytes to text stream

>>> open('temp', 'w').write('\xFF\xFE\xFD')      # Can write if embeddable in str
3
>>> open('temp', 'wb').write(b'\xFF\xFE\xFD')    # Can also write in binary mode
3

>>> open('temp', 'rb').read()                    # Can always read as binary bytes
b'\xff\xfe\xfd'

>>> open('temp', 'r').read()                     # Can't read text unless decodable!
UnicodeEncodeError: 'charmap' codec can't encode characters in position 2-3: ...
```

□□□□□□□□□□□□□□——Python 3.0□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□

□□Unicode□□

□□Unicode□□□□□□□□□□□□□□□□□□□

```
b'A\xc4B\xe8C'
>>> len(L)
5

>>> U = S.encode('utf-8')                    # 7 bytes when encoded as utf-8
>>> U
b'A\xc3\x84B\xc3\xa8C'
>>> len(U)
7
```

□□□□□□

□□□□□

```
# Decoding automatically when read

>>> open('latindata', 'r', encoding='latin-1').read()          # Decoded on input
'AÄBèC'
>>> open('utf8data', 'r', encoding='utf-8').read()             # Per encoding type
'AÄBèC'

>>> X = open('latindata', 'rb').read()                         # Manual decoding
>>> X.decode('latin-1')                                       # Not necessary
'AÄBèC'
>>> X = open('utf8data', 'rb').read()
>>> X.decode()                                                # UTF-8 is default
'AÄBèC'
```

□□□□□□

□□□□□□□Python 3.0□□□□□□□

□□□

□Python 3.0□□□BOM

□□□

```
>>> open('spam.txt', 'rb').read()                    # UTF-8 with 3-byte BOM
b'\xef\xbb\xbfspam\r\nSPAM\r\n'
>>> open('spam.txt', 'r').read()
'ï»¿spam\nSPAM\n'
>>> open('spam.txt', 'r', encoding='utf-8').read()
'\ufeffspam\nSPAM\n'
>>> open('spam.txt', 'r', encoding='utf-8-sig').read()
'spam\nSPAM\n'
```

□□□□□□“Unicode□□”□□□Notepad□□□□□□□□□□□UTF-16□□□□□□□□□□□□□□□BOM□□——□Python□□□□□"utf-16"□□BOM□□□□□□□□□□□□□□UTF-16□□□□□□□BOM□□□□"utf-16-be"□□□□□□□□□□□□BOM□

```
>>> open('spam.txt', 'rb').read()
b'\xfe\xff\x00s\x00p\x00a\x00m\x00\r\x00\n\x00S\x00P\x00A\x00M\x00\r\x00\n'
>>> open('spam.txt', 'r').read()
UnicodeEncodeError: 'charmap' codec can't encode character '\xfe' in position 1:...
>>> open('spam.txt', 'r', encoding='utf-16').read()
'spam\nSPAM\n'
>>> open('spam.txt', 'r', encoding='utf-16-be').read()
'\ufeffspam\nSPAM\n'
```

□□□□□□□□□□□□□□□Python□□□□□□Unicode□□□□□□□□□□□□□□□□□□□□□□□□UTF-8□□□BOM——"utf-8"□□□□□□□□□BOM□□"utf-8-sig"□□□□□

```
>>> open('temp.txt', 'w', encoding='utf-8').write('spam\nSPAM\n')
10
>>> open('temp.txt', 'rb').read()                          # No BOM
b'spam\r\nSPAM\r\n'

>>> open('temp.txt', 'w', encoding='utf-8-sig').write('spam\nSPAM\n')
10
>>> open('temp.txt', 'rb').read()                          # Wrote BOM
b'\xef\xbb\xbfspam\r\nSPAM\r\n'

>>> open('temp.txt', 'r').read()
'ï>>¿spam\nSPAM\n'
>>> open('temp.txt', 'r', encoding='utf-8').read()         # Keeps BOM
'\ufeffspam\nSPAM\n'
>>> open('temp.txt', 'r', encoding='utf-8-sig').read()     # Skips BOM
'spam\nSPAM\n'
```

□□□□□"utf-8"□□□□BOM□□□□BOM□□□□□□"utf-8"□"utf-8-sig"□□——□□□□□□□□□□□□□□□BOM□□□□□□□□□□□□□□□

```
>>> sys.byteorder
'little'
>>> open('temp.txt', 'w', encoding='utf-16').write('spam\nSPAM\n')
10
>>> open('temp.txt', 'rb').read()
b'\xff\xfes\x00p\x00a\x00m\x00\r\x00\n\x00S\x00P\x00A\x00M\x00\r\x00\n\x00'
>>> open('temp.txt', 'r', encoding='utf-16').read()
'spam\nSPAM\n'

>>> open('temp.txt', 'w', encoding='utf-16-be').write('\ufeffspam\nSPAM\n')
11
>>> open('spam.txt', 'rb').read()
b'\xfe\xff\x00s\x00p\x00a\x00m\x00\r\x00\n\x00S\x00P\x00A\x00M\x00\r\x00\n'
>>> open('temp.txt', 'r', encoding='utf-16').read()
'spam\nSPAM\n'
>>> open('temp.txt', 'r', encoding='utf-16-be').read()
'\ufeffspam\nSPAM\n'
```

□□□□UTF-16□□□□□□□□□BOM□□□□□□□□□□□□□"utf-16"□□□□□□□□BOM□□□□□□□□□

```
>>> open('temp.txt', 'w', encoding='utf-16-le').write('SPAM')
4
>>> open('temp.txt', 'rb').read()                # OK if BOM not present or expected
b'S\x00P\x00A\x00M\x00'
>>> open('temp.txt', 'r', encoding='utf-16-le').read()
'SPAM'
>>> open('temp.txt', 'r', encoding='utf-16').read()
UnicodeError: UTF-16 stream does not start with BOM
```

□□□□□□□□□□□□□□□□Python□□□□□□□□□□BOM□□□□□□

Python 2.6□□Unicode□□

□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□Python 2.6□□ Unicode□□□□□□□□□□□□□□□□□□□□□□□□□□□□unicode□□str□□□ codecs.open□□□□□Python 2.6□□□□□□□□□

```
C:\misc> c:\python26\python
>>> S = u'A\xc4B\xe8C'
>>> print S
AÄBèC
>>> len(S)
5
>>> S.encode('latin-1')
'A\xc4B\xe8C'
>>> S.encode('utf-8')
'A\xc3\x84B\xc3\xa8C'

>>> import codecs
>>> codecs.open('latindata', 'w', encoding='latin-1').write(S)
>>> codecs.open('utfdata', 'w', encoding='utf-8').write(S)

>>> open('latindata', 'rb').read()
'A\xc4B\xe8C'
>>> open('utfdata', 'rb').read()
'A\xc3\x84B\xc3\xa8C'

>>> codecs.open('latindata', 'r', encoding='latin-1').read()
u'A\xc4B\xe8C'
>>> codecs.open('utfdata', 'r', encoding='utf-8').read()
u'A\xc4B\xe8C'
```

Python 3.0□□□□□□□□□□□□

Python□□□□□□□□□□□□

```
C:\misc> c:\python30\python
>>> import re
>>> S = 'Bugger all down here on earth!'                    # Line of text
>>> B = b'Bugger all down here on earth!'                   # Usually from a file

>>> re.match('(.*) down (.*) on (.*)', S).groups()          # Match line to pattern
('Bugger all', 'here', 'earth!')                            # Matched substrings

>>> re.match(b'(.*) down (.*) on (.*)', B).groups()         # bytes substrings
(b'Bugger all', b'here', b'earth!')
```

Python 2.6□□□□□□□□□□□□□unicode□□□□□ASCII□□□□□str□□8□□□□□□□□□

```
C:\misc> c:\python26\python
>>> import re
>>> S = 'Bugger all down here on earth!'                    # Simple text and binary
>>> U = u'Bugger all down here on earth!'                   # Unicode text

>>> re.match('(.*) down (.*) on (.*)', S).groups()
('Bugger all', 'here', 'earth!')

>>> re.match('(.*) down (.*) on (.*)', U).groups()
(u'Bugger all', u'here', u'earth!')
```

□□bytes□str□□□□□□□□□□□□□

```
C:\misc> c:\python30\python
>>> import re
>>> S = 'Bugger all down here on earth!'
>>> B = b'Bugger all down here on earth!'

>>> re.match('(.*) down (.*) on (.*)', B).groups()
TypeError: can't use a string pattern on a bytes-like object

>>> re.match(b'(.*) down (.*) on (.*)', S).groups()
TypeError: can't use a bytes pattern on a string-like object

>>> re.match(b'(.*) down (.*) on (.*)', bytearray(B)).groups()
(bytearray(b'Bugger all'), bytearray(b'here'), bytearray(b'earth!'))

>>> re.match('(.*) down (.*) on (.*)', bytearray(B)).groups()
TypeError: can't use a string pattern on a bytes-like object
```

Struct□□□□□□□

Python□struct□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□Python 2.X□□□□□□□□□□□□□□□□□□□□bytes□bytearray□□□□□□□□□str□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python
>>> import struct
>>> B = struct.pack('>i4sh', 7, 'spam', 8)
>>> B
b'\x00\x00\x00\x07spam\x00\x08'

>>> vals = struct.unpack('>i4sh', B)
>>> vals
(7, b'spam', 8)

>>> vals = struct.unpack('>i4sh', B.decode())
TypeError: 'str' does not have the buffer interface
```

□□bytes□□□□□□□□□□□□□□□□□□□□Python 3.0□Python 2.X□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□

□□□□□□Python□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

pickle□□□□□□□

□□□□□□9□□□30□□□□□□□pickle□□□□□27□□□□□□□□□shelve□□□□□□□□□pickle□□□□□□□□□□□□□□□pickle□□□Python 3.0□□□□□□□□□bytes□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□dumps□□□□□□□□□pickle□□□□□□□□□□□□

```
C:\misc> C:\Python30\python
>>> import pickle                          # dumps() returns pickle string

>>> pickle.dumps([1, 2, 3])                # Python 3.0 default protocol=3=binary
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'

>>> pickle.dumps([1, 2, 3], protocol=0)    # ASCII protocol 0, but still bytes!
b'(lp0\nL1L\naL2L\naL3L\na.'
```

□□□□□□□□□pickle□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□str□□□□□□□□□□□□□bytes——dump□□□□□□□pickle□□□□□□□□□□□□□□□:

```
>>> pickle.dump([1, 2, 3], open('temp', 'w'))     # Text files fail on bytes!
TypeError: can't write bytes to text stream       # Despite protocol value

>>> pickle.dump([1, 2, 3], open('temp', 'w'), protocol=0)
TypeError: can't write bytes to text stream

>>> pickle.dump([1, 2, 3], open('temp', 'wb'))    # Always use binary in 3.0

>>> open('temp', 'r').read()
UnicodeEncodeError: 'charmap' codec can't encode character '\u20ac' in ...
```

□□pickle□□□□□□□□Unicode□□□□□□□□□□□□□□——□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□pickle□□□

```
>>> pickle.dump([1, 2, 3], open('temp', 'wb'))
>>> pickle.load(open('temp', 'rb'))
[1, 2, 3]
>>> open('temp', 'rb').read()
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□pickle□□□□□□□□□□0□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python26\python
>>> import pickle
>>> pickle.dumps([1, 2, 3])                    # Python 2.6 default=0=ASCII
'(lp0\nI1\naI2\naI3\na.'

>>> pickle.dumps([1, 2, 3], protocol=1)
']q\x00(K\x01K\x02K\x03e.'

>>> pickle.dump([1, 2, 3], open('temp', 'w'))  # Text mode works in 2.6
>>> pickle.load(open('temp'))
[1, 2, 3]
>>> open('temp').read()
'(lp0\nI1\naI2\naI3\na.'
```

□□□□□□□□□□□□□□□□□□□

```
>>> import pickle
>>> pickle.dump([1, 2, 3], open('temp', 'wb'))    # Version neutral
>>> pickle.load(open('temp', 'rb'))               # And required in 3.0
[1, 2, 3]
```

□□□□□□□□□□□□□Python□□□pickle□unpikle□□□□□□□□□pickle□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3□□pickle□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□pickle□□□□□□□

XML□□□□

XML□□□□

```
<books>
    <date>2009</date>
    <title>Learning Python</title>
    <title>Programming Python</title>
    <title>Python Pocket Reference</title>
    <publisher>O'Reilly Media</publisher>
</books>
```

□□□

□□□□□□□□□□□□□□□□□□□□□□□□DOM□□□□□□□□□□□XML□□□DOM□XML□□□□□□□□□□□□□□□□□□□

```
            self.inTitle = False
        def startElement(self, name, attributes):
            if name == 'title':
                self.inTitle = True
        def characters(self, data):
            if self.inTitle:
                print(data)
        def endElement(self, name):
            if name == 'title':
                self.inTitle = False

    import xml.sax
    parser = xml.sax.make_parser()
    handler = BookHandler()
    parser.setContentHandler(handler)
    parser.parse('mybooks.xml')
```

□□□□□□□□□□□□etree□□□ElementTree□□□□□□□□□□□□XML DOM □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□XML□□□□□□□□XML □□□□□□□□□□□□API□□□□□□□□□□□□

```
    # File etreeparse.py

    from xml.etree.ElementTree import parse
    tree = parse('mybooks.xml')
    for E in tree.findall('title'):
        print(E.text)
```

□Python 2.6□Python 3.0□□□□□□□□□□4□□□□□□□□□□□□

```
C:\misc> c:\python26\python domparse.py
Learning Python
Programming Python
Python Pocket Reference

C:\misc> c:\python30\python domparse.py
Learning Python
Programming Python
Python Pocket Reference
```

□□□□□□□□□Python 2.6□□□□□□□□□□□□□unicode□□□□□□□□

Python 3.0□□□□str□□□□□□□□□□□□□Unicode□□□□□□ASCII□□□□□

```
C:\misc> c:\python30\python
>>> from xml.dom.minidom import parse, Node
>>> xmltree = parse('mybooks.xml')
>>> for node in xmltree.getElementsByTagName('title'):
...     for node2 in node.childNodes:
...         if node2.nodeType == Node.TEXT_NODE:
...             node2.data
...
```

```
'Learning Python'
'Programming Python'
'Python Pocket Reference'

C:\misc> c:\python26\python
>>> ...same code...
...
u'Learning Python'
u'Programming Python'
u'Python Pocket Reference'
```

□□□□XML□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□

□□□□

□□□□□Python 3.0□Python 2.6□□□□□□□Unicode□□□□□□□

□□□□

1.Python 3.0□□□□□□□□□□□□□□□□□□□□□□

2.Python 2.6□□□□□□□□□□□□□□□□□□□□□□

3.Python 2.6□Python 3.0□□□□□□□□□□□□□□□□

4.Python 3.0□□□□□□□□□□□□□□□□□□

5.□Python 3.0□□□□□□□ASCII Unicode□□□□□□□□□□□□

6.Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

7.□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□

□□□□

1.Python 3.0□3□□□□□□□□str□□□Unicode□□□□□□ASCII□□bytes□□□□□□□□

6.□Python 3.0□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□□□□□□□□□

3.0□□□□□□□□□□□□□□□□□□□□re□struct□pickle□xml□□□□□□□□□□□□□□□□
Python 3.0□Python 2.6□□□
□Python 3.0□str□bytes□□Python 2.6□str□□□□□□□□□□□□□□□□□□□□
Unicode□□□□□□□□□□Python 2.6□□□□unicode□codecs.open()□□
□Python 3.0□str□open□□□□□□□□□□□□□□□□□□□□□□□□□□bytes□□□□□□
□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□37□ □□□□

□□□

```
                class Person:
                    def getName(self):
                        if not valid():


        raise TypeError('cannot fetch name')
    else:
        return self.name.transform()
    def setName(self, value):
         if not valid(value):
            raise TypeError('cannot change name')
        else:
            self.name = transform(value)

    person = Person()
    person.getName()
    person.setName('value')
```

□□□

□□□

□□□

□□

□□□□□□□□□□□□□□□□□get□set□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□fget□□□□□□□□□□□□□□fget□□□□□□□□□□□□□□□None□□fget□□□□□□□□□□□□□□fset□fdel□□□□□□□□□□None□□

□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□

□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□31□□□□□□□□□□□□□□□□□□□

□□Python 2.6□property□□□□getter□setter□deleter□□□□

□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□

□□□

□□□□

□□□

□□□□□□□□□□□□□□□□X.attr□□□□Python□□□□Descriptor□□__get__□□□Subject.attr□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□

```
>>> class Descriptor(object):
...     def __get__(self, instance, owner):
...         print(self, instance, owner, sep='\n')
...
>>> class Subject:
...     attr = Descriptor()              # Descriptor instance is class attr
...
>>> X = Subject()

>>> X.attr
<__main__.Descriptor object at 0x0281E690>
<__main__.Subject object at 0x028289B0>
<class '__main__.Subject'>

>>> Subject.attr
<__main__.Descriptor object at 0x0281E690>
None
<class '__main__.Subject'>
```

□□□□□□□□□□□□□□□□□□__get__□□□□□□□□□□X.attr□□□□□□□□□□□□□□□□□□□□□□Subject.attr□□□□□□__get__□□

```
X.attr -> Descriptor.__get__(Subject.attr, X, Subject)
```

□□□□□□□□□□None□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__set__□□□□□□□□□□□□□□□□□□□

```
>>> class D:
...     def __get__(*args): print('get')
...     def __set__(*args): raise AttributeError('cannot set')
...
>>> class C:
...     a = D()
...
>>> X = C()
>>> X.a                                  # Routed to C.a.__get__
get
>>> X.a = 99                             # Routed to C.a.__set__
AttributeError: cannot set
```

□□□□□□□□□□□□□□□__delete__□□□□□□__del__□□□□□□□□□□□□□□□

□□□□□

□□□□

□□□

□□□

□□□□□□□□□□□□Name□□□Person□□□□□□□□□□□□

□□□□□

□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
class DescState:                                    # Use descriptor state
    def __init__(self, value):
        self.value = value
    def __get__(self, instance, owner):             # On attr fetch
        print('DescState get')
        return self.value * 10
    def __set__(self, instance, value):             # On attr assign
        print('DescState set')
        self.value = value

# Client class

class CalcAttrs:
    X = DescState(2)                                # Descriptor class attr
    Y = 3                                           # Class attr
    def __init__(self):
        self.Z = 4                                  # Instance attr

obj = CalcAttrs()
print(obj.X, obj.Y, obj.Z)                          # X is computed, others are not
obj.X = 5                                           # X assignment is intercepted
obj.Y = 6
obj.Z = 7
print(obj.X, obj.Y, obj.Z)
```

□□□□□Value□□□□□

□□□

```
DescState get
InstState get
20 300 4
DescState set
InstState set
DescState get
InstState get
50 600 7
```

□□□

□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□——property□□□□□□□□□□□□□□□□□□□

□□Property□□□

__getattr__和__getattribute__

□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□__getattr__和__getattribute__□□□□□□□□□□□□□□□□□□□

__getattr__和__getattribute__□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□

□□□□□□□□self□□□□□□□□□□□□name□□□□□□□□□□□□□□□□value□□□□□□□□□□□□□get□□□□□□□□□□□□□□□□□□□□□□□□□□□□□None□□□□□□□□□□□□□□□□□□□

□□□__getattr__□□□□□□□□□□□□□□□□□□□

```
def __setattr__(self, name, value):
    self.__dict__['other'] = value                    # Use atttr dict to avoid me
```

□□□□□□□□□□□□□□□__setattr__□□□□□□□□□□□□□□□

□□□□□

□□□□

```
            print('fetch...')
            return self._name                    # Does not loop: real attr
        else:                                    # Others are errors
            raise AttributeError(attr)

    def __setattr__(self, attr, value):          # On [obj.any = value]
        if attr == 'name':
           print('change...')
           attr = '_name'                        # Set internal name
        self.__dict__[attr] = value              # Avoid looping here

    def __delattr__(self, attr):                 # On [del obj.any]
        if attr == 'name':
           print('remove...')
           attr = '_name'                        # Avoid looping here too
        del self.__dict__[attr]                  # but much less common

bob = Person('Bob Smith')                        # bob has a managed attribute
print(bob.name)                                  # Runs __getattr__
bob.name = 'Robert Smith'                        # Runs __setattr__
print(bob.name)
del bob.name                                     # Runs __delattr__

print('-'*20)
sue = Person('Sue Jones')                        # sue inherits property too
print(sue.name)
#print(Person.name.__doc__)                      # No equivalent here
```

□□□__init__□□□□□□□□□□□□□□□__setattr__□□□□□□□□□□□□□□□

```
fetch...
Bob Smith
change...
fetch...
Robert Smith
remove...
--------------------
fetch...
Sue Jones
```

□□□

□□□

□□□□

□□□

□□□□□□□□□□□__getattribute__□□□__getattr__□□□□□□□□□□□□□□□□□□__getattribute__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__dict__□□□□__setattr__□□□□□□□□□□□□

```
class AttrSquare:
    def __init__(self, start):
        self.value = start                              # Triggers __setattr__!

    def __getattribute__(self, attr):                   # On all attr fetches
        if attr == 'X':
            return self.value ** 2                      # Triggers __getattribute__ again!
        else:
            return object.__getattribute__(self, attr)

    def __setattr__(self, attr, value):                 # On all attr assignments
        if attr == 'X':
            attr = 'value'
        object.__setattr__(self, attr, value)
```

□□□□□□□□□□□□□□□□□□□

□□□

__getattr__和__getattribute__的区别

下面通过__getattr__和__getattribute__的例子来说明这两个方法的区别，这个例子中有3个属性——attr1是类的属性，attr2是实例的属性，attr3是不存在的属性，通过这个例子来说明这两个方法的区别。

```
class GetAttr:
    attr1 = 1
    def __init__(self):
        self.attr2 = 2
    def __getattr__(self, attr):                # On undefined attrs only
        print('get: ' + attr)                    # Not attr1: inherited from class
        return 3                                 # Not attr2: stored on instance

X = GetAttr()


print(X.attr1)
print(X.attr2)
print(X.attr3)

print('-'*40)

class GetAttribute(object):                      # (object) needed in 2.6 only
    attr1 = 1
    def __init__(self):
        self.attr2 = 2
    def __getattribute__(self, attr):           # On all attr fetches
        print('get: ' + attr)                    # Use superclass to avoid looping here
        if attr == 'attr3':
            return 3
        else:
            return object.__getattribute__(self, attr)

X = GetAttribute()
print(X.attr1)
print(X.attr2)
print(X.attr3)
```

□□□□__getattr__□□□□□attr3□□□□□

2 dynamically computed attributes with properties

```
class Powers:
    def __init__(self, square, cube):
        self._square = square                    # _square is the base value
        self._cube = cube                        # square is the property name

    def getSquare(self):
        return self._square ** 2
    def setSquare(self, value):
        self._square = value
    square = property(getSquare, setSquare)

    def getCube(self):
        return self._cube ** 3
    cube = property(getCube)

X = Powers(3, 4)
print(X.square)       # 3 ** 2 = 9
print(X.cube)         # 4 ** 3 = 64
X.square = 5
print(X.square)       # 5 ** 2 = 25
```

□□□

```
# Same, but with descriptors

class DescSquare:
    def __get__(self, instance, owner):
        return instance._square ** 2
    def __set__(self, instance, value):
        instance._square = value

class DescCube:
    def __get__(self, instance, owner):
        return instance._cube ** 3

class Powers:                                   # Use (object) in 2.6
    square = DescSquare()
    cube = DescCube()
    def __init__(self, square, cube):
        self._square = square                   # "self.square = square" works too,
        self._cube = cube                       # because it triggers desc __set__!

X = Powers(3, 4)
print(X.square)       # 3 ** 2 = 9
print(X.cube)         # 4 ** 3 = 64
X.square = 5
print(X.square)       # 5 ** 2 = 25
```

□□□__getattr__□□□

```
# Same, but with generic __getattr__ undefined attribute interception

class Powers:
    def __init__(self, square, cube):
        self._square = square
        self._cube =   cube
    def __getattr__(self, name):
        if name == 'square':
            return self._square ** 2
        elif name == 'cube':
            return self._cube ** 3
        else:
            raise TypeError('unknown attr:' + name)
        def __setattr__(self, name, value):
            if name == 'square':
                self.__dict__['_square'] = value
            else:
                self.__dict__[name] = value

X = Powers(3, 4)
print(X.square)       # 3 ** 2 = 9
print(X.cube)         # 4 ** 3 = 64
X.square = 5
print(X.square)       # 5 ** 2 = 25
```

□□□□□□□□□□__getattribute__□□□□□□□□□□

```
# Same, but with generic __getattribute__ all attribute interception

class Powers:
    def __init__(self, square, cube):
        self._square = square
        self._cube = cube
    def __getattribute__(self, name):
        if name == 'square':
            return object.__getattribute__(self, '_square') ** 2
        elif name == 'cube':
            return object.__getattribute__(self, '_cube') ** 3
        else:
            return object.__getattribute__(self, name)
     def __setattr__(self, name, value):
        if name == 'square':
           self.__dict__['_square'] = value
        else:
           self.__dict__[name] = value

X = Powers(3, 4)
print(X.square)       # 3 ** 2 = 9
print(X.cube)         # 4 ** 3 = 64
X.square = 5

print(X.square)       # 5 ** 2 = 25
```

□□□

□□□□□□□□□

□□□__getattr__□__getattribute__□□□□□□□□

□□□

```
class GetAttr:
    eggs = 88                           # eggs stored on class, spam on instance
    def __init__(self):
        self.spam = 77
    def __len__(self):                  # len here, else __getattr__ called with __len__
        print('__len__: 42')
        return 42
    def __getattr__(self, attr):        # Provide __str__ if asked, else dummy func
        print('getattr: ' + attr)
        if attr == '__str__':
            return lambda *args: '[Getattr str]'
        else:
            return lambda *args: None

class GetAttribute(object):             # object required in 2.6, implied in 3.0
    eggs = 88                           # In 2.6 all are isinstance(object) auto
    def __init__(self):                 # But must derive to get new-style tools,
        self.spam = 77                  # incl __getattribute__, some __X__ defaults
    def __len__(self):
        print('__len__: 42')
        return 42
    def __getattribute__(self, attr):
        print('getattribute: ' + attr)
        if attr == '__str__':
            return lambda *args: '[GetAttribute str]'
        else:
            return lambda *args: None

for Class in GetAttr, GetAttribute:
    print('\n' + Class.__name__.ljust(50, '='))

    X = Class()
    X.eggs                      # Class attr
    X.spam                      # Instance attr
    X.other                     # Missing attr
    len(X)                      # __len__ defined explicitly

    try:                        # New-styles must support [], +, call directly: redefine
        X[0]                    # __getitem__?
    except:
        print('fail []')
```

```
try:
    X + 99                          # __add__?
except:
    print('fail +')

try:
    X()                             # __call__? (implicit via built-in)
except:
    print('fail ()')
X.__call__()                        # __call__? (explicit, not inherited)

print(X.__str__())                  # __str__? (explicit, inherited from type)
print(X)                            # __str__? (implicit via built-in)
```

□Python 2.6□□□□□□□□__getattr__□□□□□□□□

```
C:\misc> c:\python26\python getattr.py

GetAttr=========================================
getattr: other
__len__: 42
getattr: __getitem__
getattr: __coerce__
getattr: __add__
getattr: __call__
getattr: __call__
getattr: __str__
[Getattr str]
getattr: __str__
[Getattr str]

GetAttribute====================================
getattribute: eggs
getattribute: spam
getattribute: other
__len__: 42
fail []
fail +
fail ()
getattribute: __call__
getattribute: __str__
[GetAttribute str]
<__main__.GetAttribute object at 0x025EA1D0>
```

□□□□Python 2.6□□□□□__getattr__□□□□__call__□__str__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__getattribute__□□□□□□□□□

□□□__getattribute__□□□Python 2.6□□□□Python 3.0□□□□□□□□□Python 2.6□□□□□□□□□□object□□□□□□□□□□□□□□□□□□□□□□□□object□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□

□□□□Python 3.0□□□□□□□□__getattr__□□□□□□——□□□□□□□□□□□□□□□□□□□

```
C:\misc> c:\python30\python getattr.py

GetAttr=========================================
getattr: other
__len__: 42
fail []
fail +
fail ()
getattr: __call__
<__main__.GetAttr object at 0x025D17F0>
<__main__.GetAttr object at 0x025D17F0>

GetAttribute====================================
getattribute: eggs
getattribute: spam
getattribute: other
__len__: 42
fail []
fail +
fail ()
getattribute: __call__
getattribute: __str__
[GetAttribute str]
<__main__.GetAttribute object at 0x025D1870>
```

□□

·□Python 3.0□□__str__□□□□□□□□__getattr__□□□□□□□□□□□□□□□□□□□

·__str__□□□□□□□□__getattribute__□□□□□□□□

□□□__str__□□□□□□□
__getattribute__□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□□□Manager

□27□□□□□□□□□□□□□□□Manager□□□□□□□□□□□□□□□

□□□

```
C:\misc> c:\python30\python person.py
Jones
[Person: Sue Jones, 110000]
Jones
<__main__.Manager object at 0x02A5AE30>
```

□□□□□□□□□□□□□__str__□□□□Python 2.6□□□□□__getattr__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__str__□□□□□□□□□□

```
C:\misc> c:\python26\python person.py
Jones
[Person: Sue Jones, 110000]
Jones
[Person: Tom Jones, 60000]
```

□□□__getattribute__□□□□□□□Python 3.0□□□□——□__getattr__□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
# Replace __getattr_ with __getattribute__

class Manager:                                          # Use (object) in 2.6
    def __init__(self, name, pay):
        self.person = Person(name, 'mgr', pay)          # Embed a Person object
    def giveRaise(self, percent, bonus=.10):
        self.person.giveRaise(percent + bonus)          # Intercept and delegate
    def __getattribute__(self, attr):
        print('**', attr)
        if attr in ['person', 'giveRaise']:
            return object.__getattribute__(self, attr)  # Fetch my attrs
        else:
            return getattr(self.person, attr)           # Delegate all others
```

□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□Manager□□□□□□□□□□□□□□__str__□□□□□□□□□□□□□□□□□□□

```
# Code __getattribute__ differently to minimize extra calls

class Manager:
    def __init__(self, name, pay):
        self.person = Person(name, 'mgr', pay)
    def __getattribute__(self, attr):
        print('**', attr)
        person = object.__getattribute__(self, 'person')
        if attr == 'giveRaise':
            return lambda percent: person.giveRaise(percent+.10)
        else:
            return getattr(person, attr)
        def __str__(self):
            person = object.__getattribute__(self, 'person')
            return str(person)
```

□□□

□□□

□□□□□□□

□□□□□□

□□□□□□□□□□□□name□age□acct□□□□□□□□□□□□addr□□□□□□□□□□□remain□□□□□□□□□□□□

```
        self.__age = value
age = property(getAge, setAge)

def getAcct(self):
    return self.__acct[:-3] + '***'
def setAcct(self, value):
    value = value.replace('-', '')
    if len(value) != self.acctlen:
        raise TypeError('invald acct number')
    else:
        self.__acct = value
acct = property(getAcct, setAcct)

def remainGet(self):                        # Could be a method, not attr
    return self.retireage - self.age        # Unless already using as attr
remain = property(remainGet)
```

self□□□□

□□□□

```
bob = CardHolder('1234-5678', 'Bob Smith', 40, '123 main st')
print(bob.acct, bob.name, bob.age, bob.remain, bob.addr, sep=' / ')
bob.name = 'Bob Q. Smith'
bob.age = 50
bob.acct = '23-45-67-89'
print(bob.acct, bob.name, bob.age, bob.remain, bob.addr, sep=' / ')

sue = CardHolder('5678-12-34', 'Sue Jones', 35, '124 main st')
print(sue.acct, sue.name, sue.age, sue.remain, sue.addr, sep=' / ')
try:
    sue.age = 200
except:
    print('Bad age for Sue')

try:
    sue.remain = 5
except:
    print("Can't set sue.remain")

try:
    sue.acct = '1234567'
except:
    print('Bad acct for Sue')
```

□□□□□□self□□□□□□

□□□□□□□

□□□□□□

```
class CardHolder:
    acctlen = 8                                 # Class data
    retireage = 59.5

    def __init__(self, acct, name, age, addr):
        self.acct = acct                        # Instance data
        self.name = name                        # These trigger __set__ calls too
    self.age = age                              # __X not needed: in descriptor
    self.addr = addr                            # addr is not managed
                                                # remain has no data
class Name:


       def __get__(self, instance, owner):      # Class names: CardHolder locals
           return self.name
       def __set__(self, instance, value):
           value = value.lower().replace(' ', '_')
           self.name = value
   name = Name()

   class Age:
       def __get__(self, instance, owner):
           return self.age                      # Use descriptor data
       def __set__(self, instance, value):
           if value < 0 or value > 150:
               raise ValueError('invalid age')
           else:
               self.age = value
   age = Age()

   class Acct:
       def __get__(self, instance, owner):
           return self.acct[:-3] + '***'
       def __set__(self, instance, value):
           value = value.replace('-', '')
           if len(value) != instance.acctlen:   # Use instance class data
               raise TypeError('invald acct number')
           else:
               self.acct = value
   acct = Acct()

   class Remain:
       def __get__(self, instance, owner):
           return instance.retireage - instance.age   # Triggers Age.__get__
       def __set__(self, instance, value):
           raise TypeError('cannot set remain')       # Else set allowed here
   remain = Remain()
```

□□__getattr__□□□

□□□□□□□□□□□__getattr__□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□addr□□□□□□□□□□□□□□□□□□□

□□__getattribute__□□

□□□□□□□□__getattribute__□□□□□□□□

```
class CardHolder:
    acctlen = 8                                 # Class data
    retireage = 59.5

    def __init__(self, acct, name, age, addr):
        self.acct = acct                        # Instance data
        self.name = name                        # These trigger __setattr__ too
        self.age = age                          # acct not mangled: name tested
        self.addr = addr                        # addr is not managed
                                                # remain has no data
    def __getattribute__(self, name):
        superget = object.__getattribute__              # Don't loop: one level up
        if name == 'acct':                              # On all attr fetches
            return superget(self, 'acct')[:-3] + '***'
        elif name == 'remain':
            return superget(self, 'retireage') - superget(self, 'age')
        else:
            return superget(self, name)                 # name, age, addr: stored

    def __setattr__(self, name, value):
        if name == 'name':                              # On all attr assignments
            value = value.lower().replace(' ', '_')     # addr stored directly
        elif name == 'age':
            if value < 0 or value > 150:
                raise ValueError('invalid age')
        elif name == 'acct':
            value = value.replace('-', '')
            if len(value) != self.acctlen:
                raise TypeError('invald acct number')
       elif name == 'remain':
           raise TypeError('cannot set remain')
       self.__dict__[name] = value      # Avoid loops, orig names
```

□□□

□□□□

□□□□□□Python□□□□□□□□□□□□□□□□□□□□□__getattr__□__getattribute__□□□□□□□□□□□□□□□□□□□

□□□□

1.__getattr__□__getattribute__□□□□□

2.□□□□□□□□□□□□

3.□□□□□□□□□□□□

4.__getattr__□__getattribute__□□□□□□□□□□□□□□□□□□□□□□□□□□□□

5.□□□□□□□□□□□□□□□□□□□□

□□□□

1.__getattr__□□□__getattribute__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__getattr__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__getattribute__□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__getattr__□__getattribute__□□□□□□□□□□□□□□□□□__setattr__□□□□□□□□□□__getattr__□__getattribute__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

5.□□□□□□□Python□□□□□Monty Python's Flying Circus□□□□□

```
An argument is a connected series of statements intended to establish a
proposition.
No it isn't.
Yes it is! It's not just contradiction.
Look, if I argue with you, I must take up a contrary position.
Yes, but that's not just saying "No it isn't."
Yes it is!
No it isn't!
Yes it is!
No it isn't. Argument is an intellectual process. Contradiction is just
the automatic gainsaying of any statement the other person makes.
(short pause)
No it isn't.
It is.
Not at all.
Now look...
```

□38□　□□□

□□□□□□□□□□□□□□□□31□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□@decorator□□□□□□□□□□□37□□□□□□□□□□□□□□□□□property□□□□□□□□□□□□□□□□28□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

·□□□

□□□□□□

□□□□□

□□□□□□□□□

□□□□□□□□

□□□□□□□□□

□□□□□Python□□□□□□□□

□□□

□□□□

□□□

□□□

□□□□□□□□□□def□□□□□□□□□□

```
    # Save or use function F
    # Return a different callable: nested def, class with __call__, etc.

@decorator
def func(): ...                    # func = decorator(func)
```

□□□

```
class decorator:
    def __init__(self, func):              # On @ decoration
        self.func = func
    def __call__(self, *args):             # On wrapped function call
        # Use self.func and args
        # self.func(*args) calls original function

@decorator
def func(x, y):                            # func = decorator(func)
    ...                                    # func is passed to __init__

func(6, 7)                                 # 6, 7 are passed to __call__'s *args
```

□□□□□□□□func□□□□□□□□□□□□□□□□□□□□□□□□__call__□□□□□□□□□□□□__call__□□□□□□□□□□func□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□__call__□□□□□□□□□□□□□□□self□□□□□□□□□□□□C□□□□□□□□□□□*args□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□□□□□□

□□□

```
def decorator(cls):                              # On @ decoration
    class Wrapper:
        def __init__(self, *args):               # On instance creation
            self.wrapped = cls(*args)
        def __getattr__(self, name):             # On attribute fetch
            return getattr(self.wrapped, name)
    return Wrapper

@decorator
class C:                                         # C = decorator(C)
    def __init__(self, x, y):                    # Run by Wrapper.__init__
        self.attr = 'spam'

x = C(6, 7)                                      # Really calls Wrapper(6, 7)
print(x.attr)                                    # Runs Wrapper.__getattr__, prints "spam"
```

□□□

```
class Decorator:
    def __init__(self, C):                          # On @ decoration
        self.C = C
    def __call__(self, *args):                      # On instance creation
        self.wrapped = self.C(*args)
        return self
    def __getattr__(self, attrname):                # On atrribute fetch
        return getattr(self.wrapped, attrname)

@Decorator
class C: ...                                        # C = Decorator(C)

x = C()
y = C()                                             # Overwrites x!
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□Decorator□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□

□□□

```
def d1(F): return F
def d2(F): return F
def d3(F): return F

@d1
@d2
@d3
```

```
def func():                          # func = d1(d2(d3(func)))
    print('spam')

func()                               # Prints "spam"
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```
def d1(F): return lambda: 'X' + F()
def d2(F): return lambda: 'Y' + F()
def d3(F): return lambda: 'Z' + F()

@d1
@d2
@d3
def func():                 # func = d1(d2(d3(func)))
    return 'spam'

print(func())               # Prints "XYZspam"
```

□□□□□□□□lambda□□□□□□□□

□□□□□

□□□□

```
# Save or use A, B
def actualDecorator(F):
    # Save or use function F
    # Return a callable: nested def, class with __call__, etc.
    return callable
return actualDecorator
```

□□□

□□□□□□□□□

□□□□□□□□

□□□□□□□

□□□□□□

□□□

```
calls = 0
def tracer(func, *args):
    global calls
    calls += 1
    print('call %s to %s' % (calls, func.__name__))
    func(*args)

def spam(a, b, c):
    print(a, b, c)

>>> spam(1, 2, 3)              # Normal non-traced call: accidental?
1 2 3

>>> tracer(spam, 1, 2, 3)      # Special traced call without decorators
call 1 to spam
1 2 3
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□@□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□□□

□□□

```
calls = 0
def tracer(func):                          # State via enclosing scope and global
    def wrapper(*args, **kwargs):          # Instead of class attributes
        global calls                       # calls is global, not per-function
        calls += 1
        print('call %s to %s' % (calls, func.__name__))
        return func(*args, **kwargs)
     return wrapper

@tracer
def spam(a, b, c):                 # Same as: spam = tracer(spam)
    print(a + b + c)


@tracer
def eggs(x, y):                    # Same as: eggs = tracer(eggs)
    print(x ** y)

spam(1, 2, 3)                      # Really calls wrapper, bound to func
spam(a=4, b=5, c=6)                # wrapper calls spam

eggs(2, 16)                        # Really calls wrapper, bound to eggs
eggs(4, y=4)                       # Global calls is not per-function here!
```

□□□

```
call 1 to spam
6
call 2 to spam
15
call 3 to eggs
65536
call 4 to eggs
256
```

□□□□□□nonlocal

□□□□□

```
eggs(2, 16)                    # Really calls wrapper, bound to eggs
eggs(4, y=4)                   # Nonlocal calls _is_ not per-function here
```

□□□

```
def tracer(func):                          # State via enclosing scope and func attr
    def wrapper(*args, **kwargs):          # calls is per-function, not global
        wrapper.calls += 1
        print('call %s to %s' % (wrapper.calls, func.__name__))
        return func(*args, **kwargs)
    wrapper.calls = 0
    return wrapper
```

□□□□□□□□□□□□□□□□□□wrapper□□□□□□tracer□□□□□□□□□□□□□□□□□□□wrapper.calls□□□□□□□□□□wrapper□□□□□□□□□nonlocal□□□

□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□nonlocal□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□tracer□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□self□□□□□□□□□□□□*args□□□□□□□□□□□□□□□□□□□□□□□□□□□□tracer□□□□□□□□□□□□□□self□□□□□□□□□□□□□□□□□□□

□□27□□□□□□□□□□□□Person□□□□□□□

```
class Person:
    def __init__(self, name, pay):
        self.name = name
        self.pay = pay

    @tracer
    def giveRaise(self, percent):                    # giveRaise = tracer(giverRaise)
    self.pay *= (1.0 + percent)

    @tracer
    def lastName(self):                              # lastName = tracer(lastName)
    return self.name.split()[-1]


bob = Person('Bob Smith', 50000)                     # tracer remembers method funcs
bob.giveRaise(.25)                                   # Runs tracer.__call__(???, .25)
print(bob.lastName())                                # Runs tracer.__call__(???)
```

□□□□□□□□□□□tracer□□__call__□□□self——□□□□tracer□□□□□□□□Person□□□□□□□□□□□□□□□□□□□□□□□□□□□□tracer□□□□□□□□□Person□□□□□□□□□□□□□□□□□□□self□□□tracer□□□□□□□□tracer□□□□□□□□□□□□□□□□□□□

□□__call__□□self□□□tracer□□□Person□□□□□□□

```
<__main__.tracer object at 0x02D6AD90> (0.25,) {}
call 1 to giveRaise
Traceback (most recent call last):
  File "C:/misc/tracer.py", line 56, in <module>
    bob.giveRaise(.25)
  File "C:/misc/tracer.py", line 9, in __call__
    return self.func(*args, **kwargs)
TypeError: giveRaise() takes exactly 2 positional arguments (1 given)
```

□□□□□□□□□□□□□□□□□□□

A decorator for both functions and methods

```
def tracer(func):                           # Use function, not class with __call__
    calls = 0                               # Else "self" is decorator instance only!
    def onCall(*args, **kwargs):
        nonlocal calls
        calls += 1
        print('call %s to %s' % (calls, func.__name__))
        return func(*args, **kwargs)
    return onCall

# Applies to simple functions

@tracer
def spam(a, b, c):                          # spam = tracer(spam)
    print(a + b + c)                        # onCall remembers spam

spam(1, 2, 3)                               # Runs onCall(1, 2, 3)
spam(a=4, b=5, c=6)

# Applies to class method functions too!

class Person:
    def __init__(self, name, pay):
        self.name = name
        self.pay = pay

    @tracer
    def giveRaise(self, percent):           # giveRaise = tracer(giverRaise)
        self.pay *= (1.0 + percent)         # onCall remembers giveRaise

    @tracer
    def lastName(self):                     # lastName = tracer(lastName)
        return self.name.split()[-1]

print('methods...')
bob = Person('Bob Smith', 50000)
sue = Person('Sue Jones', 100000)
print(bob.name, sue.name)
sue.giveRaise(.10)                          # Runs onCall(sue, .10)
print(sue.pay)
print(bob.lastName(), sue.lastName())       # Runs onCall(bob), lastName in scopes
```

□□□□□□□□□□□□□□□

```
call 1 to spam
6
call 2 to spam
15
methods...
Bob Smith Sue Jones
call 1 to giveRaise
110000.0
call 1 to lastName
call 2 to lastName
Smith Jones
```

□□□□□□□□□

□□□□□□□□

□□□□□□□□__set__□__del__□□□□□□□□

□□□□tracer.__get__□□□Person□□giveRaise□□□□

□□□

□□□□

□□□

```
import time

class timer:
    def __init__(self, func):
        self.func   = func
        self.alltime = 0
    def __call__(self, *args, **kargs):
        start = time.clock()
        result = self.func(*args, **kargs)
        elapsed = time.clock() - start
        self.alltime += elapsed
        print('%s: %.5f, %.5f' % (self.func.__name__, elapsed, self.alltime))
        return result

@timer
def listcomp(N):
    return [x * 2 for x in range(N)]

@timer
def mapcall(N):
    return map((lambda x: x * 2), range(N))

result = listcomp(5)                    # Time for this call, all calls, return value
listcomp(50000)
listcomp(500000)
listcomp(1000000)
print(result)
print('allTime = %s' % listcomp.alltime)        # Total time for all listcomp calls

print('')
result = mapcall(5)
mapcall(50000)
mapcall(500000)
mapcall(1000000)
print(result)
print('allTime = %s' % mapcall.alltime)         # Total time for all mapcall calls

print('map/comp = %s' % round(mapcall.alltime / listcomp.alltime, 3))
```

□□□

□Python 2.6□□□□□□□□□□□□□□self□□□□□□□□□□

```
listcomp: 0.00002, 0.00002
listcomp: 0.00910, 0.00912
listcomp: 0.09105, 0.10017
listcomp: 0.17605, 0.27622
[0, 2, 4, 6, 8]
allTime = 0.276223304917

mapcall: 0.00003, 0.00003
mapcall: 0.01363, 0.01366
mapcall: 0.13579, 0.14945
mapcall: 0.27648, 0.42593
[0, 2, 4, 6, 8]
allTime = 0.425933533452
map/comp = 1.542
```

□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□14□□□□□□map□□□□Python 3.0□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□Python 3.0□map□□□□□□□□□□□□□□□□□□□□□□□□map□□□□□□Python 3.0□□□□□□□□

□□□□□□Python 3.0□□□□□□□□□□□□□□□list(map())□□□□□□□□□□□□□□□□□□□

□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□

□□□□□□□

□□□□□□

```
import time

def timer(label='', trace=True):                # On decorator args: retain args
    class Timer:
        def __init__(self, func):               # On @: retain decorated func
            self.func = func
            self.alltime = 0


        def __call__(self, *args, **kargs):     # On calls: call original
            start = time.clock()
            result = self.func(*args, **kargs)
            elapsed = time.clock() - start
            self.alltime += elapsed
            if trace:
               format = '%s %s: %.5f, %.5f'
               values = (label, self.func.__name__, elapsed, self.alltime)
               print(format % values)
            return result
    return Timer
```

□□□□□□□□□□□□□□□Timer□□□□□□□□□□□□□□□

```
from mytools import timer

@timer(label='[CCC]==>')
def listcomp(N):                                    # Like listcomp = timer(...)(listcomp)
    return [x * 2 for x in range(N)]                # listcomp(...) triggers Timer.__call__

@timer(trace=True, label='[MMM]==>')
def mapcall(N):
    return map((lambda x: x * 2), range(N))

for func in (listcomp, mapcall):
    print('')
    result = func(5)            # Time for this call, all calls, return value
    func(50000)
    func(500000)
    func(1000000)
    print(result)
    print('allTime = %s' % func.alltime)        # Total time for all calls

print('map/comp = %s' % round(mapcall.alltime / listcomp.alltime, 3))
```

□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□map□□□□□□□□list□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□——□□

```
[CCC]==> listcomp: 0.00003, 0.00003
[CCC]==> listcomp: 0.00640, 0.00643
[CCC]==> listcomp: 0.08687, 0.09330
[CCC]==> listcomp: 0.17911, 0.27241
[0, 2, 4, 6, 8]
```

```
allTime = 0.272407666337

[MMM]==> mapcall: 0.00004, 0.00004
[MMM]==> mapcall: 0.01340, 0.01343
[MMM]==> mapcall: 0.13907, 0.15250
[MMM]==> mapcall: 0.27907, 0.43157
[0, 2, 4, 6, 8]
allTime = 0.431572169089
map/comp = 1.584
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> from mytools import timer
>>> @timer(trace=False)                    # No tracing, collect total time
... def listcomp(N):
...     return [x * 2 for x in range(N)]
...
>>> x = listcomp(5000)
>>> x = listcomp(5000)
>>> x = listcomp(5000)
>>> listcomp
<mytools.Timer instance at 0x025C77B0>
>>> listcomp.alltime
0.0051938863738243413

>>> @timer(trace=True, label='\t=>')     # Turn on tracing
... def listcomp(N):
...     return [x * 2 for x in range(N)]
...
>>> x = listcomp(5000)
        => listcomp: 0.00155, 0.00155
>>> x = listcomp(5000)
        => listcomp: 0.00156, 0.00311
>>> x = listcomp(5000)
        => listcomp: 0.00174, 0.00486
>>> listcomp.alltime
0.0048562736325408196
```

□□□

□□□□□□

□□□□□

```
@singleton                                  # Person = singleton(Person)
class Person:                               # Rebinds Person to onCall
      def __init__(self, name, hours, rate):    # onCall remembers Person
          self.name = name
          self.hours = hours
          self.rate = rate
      def pay(self):
          return self.hours * self.rate

@singleton                                  # Spam = singleton(Spam)
class Spam:                                 # Rebinds Spam to onCall
    def __init__(self, val):                # onCall remembers Spam
        self.attr = val

bob = Person('Bob', 40, 10)                 # Really calls onCall
print(bob.name, bob.pay())

sue = Person('Sue', 50, 20)                 # Same, single object
print(sue.name, sue.pay())


X = Spam(42)                                # One Person, one Spam
Y = Spam(99)
print(X.attr, Y.attr)
```

□□□□Person□Spam□□□□

□□□

□□□□□□

□□□□□

□□□□□□□Wrapper□□□□□□□

```
            print('Trace: ' + attrname)              # Catches all but own attrs
            self.fetches += 1
            return getattr(self.wrapped, attrname)   # Delegate to wrapped obj
    return Wrapper

@Tracer
class Spam:                                          # Spam = Tracer(Spam)
    def display(self):                               # Spam is rebound to Wrapper
        print('Spam!' * 8)

@Tracer
class Person:                                        # Person = Tracer(Person)
    def __init__(self, name, hours, rate):           # Wrapper remembers Person
        self.name = name
        self.hours = hours
        self.rate = rate
    def pay(self):                                   # Accesses outside class traced
        return self.hours * self.rate                # In-method accesses not traced

food = Spam()                                        # Triggers Wrapper()
food.display()                                       # Triggers __getattr__
print([food.fetches])

bob = Person('Bob', 40, 50)                          # bob is really a Wrapper
print(bob.name)                                      # Wrapper embeds a Person
print(bob.pay())

print('')
sue = Person('Sue', rate=100, hours=60)              # sue is a different Wrapper
print(sue.name)                                      # with a different Person
print(sue.pay())

print(bob.name)                                      # bob has different state
print(bob.pay())
print([bob.fetches, sue.fetches])                    # Wrapper attrs not traced
```

□□□□□□□□“□□□□□□□”□□□□□□□□

Wrapper□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
Trace: display
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
[1]
Trace: name
Bob
Trace: pay
2000
```

```
Trace: name
Sue
Trace: pay
6000
Trace: name
Bob
Trace: pay
2000
[4, 2]
```

□□□□□□□□□□□□□□□□□□□

```
>>> from tracer import Tracer          # Decorator moved to a module file

>>> @Tracer
... class MyList(list): pass           # MyList = Tracer(MyList)

>>> x = MyList([1, 2, 3])              # Triggers Wrapper()
>>> x.append(4)                        # Triggers __getattr__, append
Trace: append
>>> x.wrapped
[1, 2, 3, 4]

>>> WrapList = Tracer(list)            # Or perform decoration manually
>>> x = WrapList([4, 5, 6])            # Else subclass statement required
>>> x.append(7)
Trace: append
>>> x.wrapped
[4, 5, 6, 7]
```

□□□

□Python 3.0□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
@Tracer
class Person:                          # Person = Tracer(Person)
    def __init__(self, name):          # Wrapper bound to Person
        self.name = name

bob = Person('Bob')                    # bob is really a Wrapper
print(bob.name)                        # Wrapper embeds a Person
Sue = Person('Sue')
print(sue.name)                        # sue overwrites bob
print(bob.name)                        # OOPS: now bob's name is 'Sue'!
```

□□□□□□□□——□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□Tracer□□□□□□□□□□□□□□□__getattr__□□□□□□□□□□□□□□□

```
instances = {}
def getInstance(object):
    aClass = object.__class__
    if aClass not in instances:
        instances[aClass] = object
    return instances[aClass]

bob = getInstance(Person('Bob', 40, 10))          # Versus: bob = Person('Bob', 40, 10)
```

□□□

□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□

□□□□□□□API□□□□□□□

□□□□□□□□

□□□□□□□

```
# Registering decorated objects to an API

registry = {}
def register(obj):                          # Both class and func decorator
    registry[obj.__name__] = obj            # Add to registry
    return obj                              # Return obj itself, not a wrapper

@register
def spam(x):
    return(x ** 2)                          # spam = register(spam)

@register
def ham(x):
    return(x ** 3)

@register
class Eggs:                                 # Eggs = register(Eggs)
    def __init__(self, x):
        self.data = x ** 4
    def __str__(self):
        return str(self.data)

print('Registry:')
for name in registry:
    print(name, '=>', registry[name], type(registry[name]))

print('\nManual calls:')
print(spam(2))                              # Invoke objects manually
print(ham(2))                               # Later calls not intercepted
X = Eggs(2)
print(X)

print('\nRegistry calls:')
for name in registry:
    print(name, '=>', registry[name](3)) # Invoke from registry
```

□□□

```
Registry:
Eggs => <class '__main__.Eggs'> <class 'type'>
ham => <function ham at 0x02CFB738> <class 'function'>
spam => <function spam at 0x02CFB6F0> <class 'function'>

Manual calls:
4
8
16

Registry calls:


 Eggs => 81
 ham => 27
 spam => 9
```

□□□

```
# Augmenting decorated objects directly

>>> def decorate(func):
...     func.marked = True              # Assign function attribute for later use
...     return func
...
>>> @decorate
... def spam(a, b):
...     return a + b
...
>>> spam.marked
True

>>> def annotate(text):                # Same, but value is decorator argument
...     def decorate(func):
...         func.label = text
...         return func
...     return decorate
...
>>> @annotate('spam data')
... def spam(a, b):                     # spam = annotate(...)(spam)
...     return a + b
...
>>> spam(1, 2), spam.label
(3, 'spam data')
```

□□□

□□□“□□”□“□□”□□

□□□

□□□□□□□□□□□□□□□□□□□□

```
"""
Privacy for attributes fetched from class instances.
See self-test code at end of file for a usage example.
Decorator same as: Doubler = Private('data', 'size')(Doubler).
Private returns onDecorator, onDecorator returns onInstance,
and each onInstance instance embeds a Doubler instance.
"""

traceMe = False
def trace(*args):
    if traceMe: print('[' + ' '.join(map(str, args)) + ']')

def Private(*privates):                         # privates in enclosing scope
    def onDecorator(aClass):                    # aClass in enclosing scope
        class onInstance:                       # wrapped in instance attribute
            def __init__(self, *args, **kargs):
                self.wrapped = aClass(*args, **kargs)
            def __getattr__(self, attr):                # My attrs don't call getattr
                trace('get:', attr)                     # Others assumed in wrapped
                if attr in privates:
                    raise TypeError('private attribute fetch: ' + attr)
                else:
                    return getattr(self.wrapped, attr)
            def __setattr__(self, attr, value):         # Outside accesses
```

```
                trace('set:', attr, value)              # Others run normally
                if attr == 'wrapped':                   # Allow my attrs
                    self.__dict__[attr] = value         # Avoid looping
                elif attr in privates:
                    raise TypeError('private attribute change: ' + attr)
                else:
                    setattr(self.wrapped, attr, value)  # Wrapped obj attrs
            return onInstance                           # Or use __dict__
        return onDecorator

if __name__ == '__main__':
    traceMe = True

    @Private('data', 'size')                            # Doubler = Private(...)(Doubler)
    class Doubler:
        def __init__(self, label, start):
            self.label = label                          # Accesses inside the subject class
            self.data = start                           # Not intercepted: run normally
        def size(self):
            return len(self.data)                       # Methods run with no checking
        def double(self):                               # Because privacy not inherited
            for i in range(self.size()):
                self.data[i] = self.data[i] * 2
        def display(self):
            print('%s => %s' % (self.label, self.data))

    X = Doubler('X is', [1, 2, 3])
    Y = Doubler('Y is', [-10, -20, -30])

    # The followng all succeed
    print(X.label)                                      # Accesses outside subject class
    X.display(); X.double(); X.display()                # Intercepted: validated, delegated
    print(Y.label)
    Y.display(); Y.double()
    Y.label = 'Spam'
    Y.display()

    # The following all fail properly
    """
    print(X.size())                     # prints "TypeError: private attribute fetch: size"
    print(X.data)
    X.data = [1, 1, 1]
    X.size = lambda S: 0
    print(Y.data)
    print(Y.size())
    """
```

□traceMe□True□□□□□□□□self□□□□□□□□

```
[set: wrapped <__main__.Doubler object at 0x02B2AAF0>]
[set: wrapped <__main__.Doubler object at 0x02B2AE70>]
[get: label]


 X is
 [get: display]
 X is => [1, 2, 3]
 [get: double]
 [get: display]
 X is => [2, 4, 6]
 [get: label]
 Y is
 [get: display]
 Y is => [-10, -20, -30]
 [get: double]
 [set: label Spam]
 [get: display]
 Spam => [-20, -40, -60]
```

□□□□□□

□□□□□

□□□□□□□□□□□

□□□□□□□□□□

□□__dict__□__slots__

□□□□□__setattr__□□□□□□□□□□__dict__□□□□□□□□□□□□□□□□onInstance□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□Private□□□□□□□□□□□□Public□□□□□□□□——□□□□□□□Private□□□□□□□□□□□□

```
"""
Class decorator with Private and Public attribute declarations.
Controls access to attributes stored on an instance, or inherited
by it from its classes. Private declares attribute names that
cannot be fetched or assigned outside the decorated class, and
Public declares all the names that can. Caveat: this works in
3.0 for normally named attributes only: __X__ operator overloading
methods implicitly run for built-in operations do not trigger
either __getattr__ or __getattribute__ in new-style classes.
Add __X__ methods here to intercept and delegate built-ins.
"""

traceMe = False
def trace(*args):
    if traceMe: print('[' + ' '.join(map(str, args)) + ']')

def accessControl(failIf):
    def onDecorator(aClass):
        class onInstance:
            def __init__(self, *args, **kargs):
                self.__wrapped = aClass(*args, **kargs)
            def __getattr__(self, attr):
                trace('get:', attr)
                if failIf(attr):
                    raise TypeError('private attribute fetch: ' + attr)
                else:
                    return getattr(self.__wrapped, attr)
            def __setattr__(self, attr, value):
                trace('set:', attr, value)
                if attr == '_onInstance__wrapped':
                    self.__dict__[attr] = value
                elif failIf(attr):
                    raise TypeError('private attribute change: ' + attr)
                else:
                    setattr(self.__wrapped, attr, value)
        return onInstance
    return onDecorator

def Private(*attributes):
    return accessControl(failIf=(lambda attr: attr in attributes))


def Public(*attributes):
    return accessControl(failIf=(lambda attr: attr not in attributes))
```

□□□□□□□self□□□□□□□

Public□□□□□□□□□□□□□□□□□□□□Private□□Public□□□□□□□□

```
>>> from access import Private, Public

>>> @Private('age')                                  # Person = Private('age')(Person)
... class Person:                                    # Person = onInstance with state
...     def __init__(self, name, age):
...         self.name = name
...         self.age = age                           # Inside accesses run normally
...
>>> X = Person('Bob', 40)
>>> X.name                                           # Outside accesses validated
'Bob'
>>> X.name = 'Sue'
>>> X.name
'Sue'
>>> X.age
TypeError: private attribute fetch: age
>>> X.age = 'Tom'
TypeError: private attribute change: age

>>> @Public('name')
... class Person:
...     def __init__(self, name, age):
...         self.name = name
...         self.age = age
...
>>> X = Person('bob', 40)                            # X is an onInstance
>>> X.name                                           # onInstance embeds Person
'bob'
>>> X.name = 'Sue'
>>> X.name
'Sue'
>>> X.age
TypeError: private attribute fetch: age
>>> X.age = 'Tom'
TypeError: private attribute change: age
```

□□□□□□

□□□□□

□□□□□

□□□□

□□□□

□□□□□□□□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□

```
...         self.age = 42
...     def __str__(self):
...         return 'Person: ' + str(self.age)
...     def __add__(self, yrs):
...         self.age += yrs
...
>>> X = Person()                                  # Name validations still work
>>> X.age                                         # But 3.0 fails to delegate built-ins!
TypeError: private attribute fetch: age
>>> print(X)
<access.onInstance object at 0x025E0790>
>>> X + 10
TypeError: unsupported operand type(s) for +: 'onInstance' and 'int'
>>> print(X)
<access.onInstance object at 0x025E0790>
```

□□□□□__getattribute__□□□□□□□□□——□□□□□□□□□

```
def accessControl(failIf):
    def onDecorator(aClass):
        class onInstance:
            def __init__(self, *args, **kargs):
                self.__wrapped = aClass(*args, **kargs)

            # Intercept and delegate operator overloading methods
            def __str__(self):
                return str(self.__wrapped)
            def __add__(self, other):
                return self.__wrapped + other
            def __getitem__(self, index):
                return self.__wrapped[index]            # If needed
            def __call__(self, *args, **kargs):
                return self.__wrapped(*arg, *kargs)     # If needed
            ...plus any others needed...

            # Intercept and delegate named attributes
            def __getattr__(self, attr):
                ...
            def __setattr__(self, attr, value):
                ...

        return onInstance
    return onDecorator
```

□□□□□□□□□□□□□□□□□□□□□__str__□__add__□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□39□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 2.6□□□□□□□□□□□□□□□□□□□

□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__setattr__□□□□□□□□□□□□□□□□□□□

□□□

Python□□□□□□

□□□□□□□□□□□□□□□□□□Python□□□Private□Public□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

```
class Person:
    def giveRaise(self, percent):                # Validate with inline code
        if percent < 0.0 or percent > 1.0:
            raise TypeError, 'percent invalid'
        self.pay = int(self.pay * (1 + percent))

class Person:                                    # Validate with asserts
    def giveRaise(self, percent):
        assert percent >= 0.0 and percent <= 1.0, 'percent invalid'
        self.pay = int(self.pay * (1 + percent))
```

□□□

□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□

Python 3.0□□print□□□□□□□□□□□

```
C:\misc> C:\python30\python devtools_test.py
True
Bob Smith is 45 years old
birthday = 5/31/1963
110000
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□TypeError□□□□□□□□□□□□□□□□□□□

```
C:\misc> C:\python30\python -O devtools_test.py
False
Bob Smith is 45 years old
birthday = 5/31/1963
110000
231000
```

□□□□□□□□□□

□□□□□□□□□

```
        code    = func.__code__
        allargs = code.co_varnames[:code.co_argcount]
        funcname = func.__name__

        def onCall(*pargs, **kargs):
            # All pargs match first N expected args by position
            # The rest must be in kargs or be omitted defaults
            positionals = list(allargs)
            positionals = positionals[:len(pargs)]

            for (argname, (low, high)) in argchecks.items():
                # For all args to be checked
                if argname in kargs:
                    # Was passed by name
                    if kargs[argname] < low or kargs[argname] > high:
                        errmsg = '{0} argument "{1}" not in {2}..{3}'
                        errmsg = errmsg.format(funcname, argname, low, high)
                        raise TypeError(errmsg)

                elif argname in positionals:
                    # Was passed by position
                    position = positionals.index(argname)
                    if pargs[position] < low or pargs[position] > high:
                        errmsg = '{0} argument "{1}" not in {2}..{3}'
                        errmsg = errmsg.format(funcname, argname, low, high)
                        raise TypeError(errmsg)
                else:
                    # Assume not passed: default
                    if trace:
                        print('Argument "{0}" defaulted'.format(argname))
                return func(*pargs, **kargs)        # OK: run original call
        return onCall
    return onDecorator
```

□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

```
# File devtools_test.py
# Comment lines raise TypeError unless "python -O" used on shell command line
from devtools import rangetest

# Test functions, positional and keyword

@rangetest(age=(0, 120))                # persinfo = rangetest(...)(persinfo)
def persinfo(name, age):
    print('%s is %s years old' % (name, age))

@rangetest(M=(1, 12), D=(1, 31), Y=(0, 2009))
def birthday(M, D, Y):
    print('birthday = {0}/{1}/{2}'.format(M, D, Y))

persinfo('Bob', 40)
persinfo(age=40, name='Bob')
birthday(5, D=1, Y=1963)
```

```
#persinfo('Bob', 150)
#persinfo(age=150, name='Bob')
#birthday(5, D=40, Y=1963)

# Test methods, positional and keyword

class Person:
    def __init__(self, name, job, pay):
        self.job = job
        self.pay = pay
                                          # giveRaise = rangetest(...)(giveRaise)
    @rangetest(percent=(0.0, 1.0))        # percent passed by name or position
    def giveRaise(self, percent):
        self.pay = int(self.pay * (1 + percent))

bob = Person('Bob Smith', 'dev', 100000)
sue = Person('Sue Jones', 'dev', 100000)
bob.giveRaise(.10)
sue.giveRaise(percent=.20)
print(bob.pay, sue.pay)
#bob.giveRaise(1.10)
#bob.giveRaise(percent=1.20)

# Test omitted defaults: skipped

@rangetest(a=(1, 10), b=(1, 10), c=(1, 10), d=(1, 10))
def omitargs(a, b=7, c=8, d=9):

    print(a, b, c, d)

omitargs(1, 2, 3, 4)
omitargs(1, 2, 3)
omitargs(1, 2, 3, d=4)
omitargs(1, d=4)
omitargs(d=4, a=1)
omitargs(1, b=2, d=4)
omitargs(d=8, c=7, a=1)

#omitargs(1, 2, 3, 11)              # Bad d
#omitargs(1, 2, 11)                 # Bad c
#omitargs(1, 2, 3, d=11)            # Bad d
#omitargs(11, d=4)                  # Bad a
#omitargs(d=4, a=11)                # Bad a
#omitargs(1, b=11, d=4)             # Bad b
#omitargs(d=8, c=7, a=11)           # Bad a
```

□□□

□□□

□□□□

□□□□□□□□□□□□API□□□□□□□□□□□□

□□□API□□□□Python□□□□□□□□□□□Python 2.5□□□□□□□□□
func.__code__attribute□□□func.func_code in 2.5□□□□□□□□□□□
__code__□□□Python 2.6□□□□□□□□□□□□□□□□□□□□□□□□□□□dir□□□□□□
□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□
Python 2.6□Python 3.0□□□□□□□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

·□def□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□

□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□VS□□□□

□□□□□Python 3.0□□□□□

```
# Using decorator arguments

def rangetest(**argchecks):
    def onDecorator(func):
        def onCall(*pargs, **kargs):
            print(argchecks)
            for check in argchecks: pass                    # Add validation code here
            return func(*pargs, **kargs)
        return onCall
    return onDecorator


@rangetest(a=(1, 5), c=(0.0, 1.0))
def func(a, b, c):                                          # func = rangetest(...)(func)
    print(a + b + c)

func(1, 2, c=3)                                             # Runs onCall, argchecks in scope

# Using function annotations

def rangetest(func):
    def onCall(*pargs, **kargs):
        argchecks = func.__annotations__
        print(argchecks)
        for check in argchecks: pass                        # Add validation code here
        return func(*pargs, **kargs)
    return onCall

@rangetest
def func(a:(1, 5), b, c:(0.0, 1.0)):                        # func = rangetest(func)
    print(a + b + c)

func(1, 2, c=3)                                             # Runs onCall, annotations on func
```

□□□

□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
@typetest
def func(a: int, b, c: float, d):        # func = typetest(func)
    ...                                  # Gasp!...
```

□□□

□□□□

□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□“□□□□□□□”□□□__call__□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□

```
import time

def timer(label='', trace=True):                # On decorator args: retain args
    def onDecorator(func):                      # On @: retain decorated func
        def onCall(*args, **kargs):             # On calls: call original
            start = time.clock()                # State is scopes + func attr
            result = func(*args, **kargs)
            elapsed = time.clock() - start
            onCall.alltime += elapsed
            if trace:
                format = '%s%s: %.5f, %.5f'
                values = (label, func.__name__, elapsed, onCall.alltime)
                print(format % values)
            return result
        onCall.alltime = 0
        return onCall
    return onDecorator

# Test on functions

@timer(trace=True, label='[CCC]==>')
def listcomp(N):                            # Like listcomp = timer(...)(listcomp)
    return [x * 2 for x in range(N)]        # listcomp(...) triggers onCall

@timer(trace=True, label='[MMM]==>')
def mapcall(N):
    return list(map((lambda x: x * 2), range(N)))     # list() for 3.0 views

for func in (listcomp, mapcall):
    result = func(5)                        # Time for this call, all calls, return value
    func(5000000)
    print(result)
    print('allTime = %s\n' % func.alltime)      # Total time for all calls

# Test on methods

class Person:
    def __init__(self, name, pay):
```

```
        self.name = name
        self.pay = pay

    @timer()
    def giveRaise(self, percent):            # giveRaise = timer()(giveRaise)
        self.pay *= (1.0 + percent)          # tracer remembers giveRaise

    @timer(label='**')
    def lastName(self):                      # lastName = timer(...)(lastName)
        return self.name.split()[-1]         # alltime per class, not instance

bob = Person('Bob Smith', 50000)
sue = Person('Sue Jones', 100000)
bob.giveRaise(.10)
sue.giveRaise(.20)                           # runs onCall(sue, .10)
print(bob.pay, sue.pay)
print(bob.lastName(), sue.lastName())        # runs onCall(bob), remembers lastName
print('%.5f %.5f' % (Person.giveRaise.alltime, Person.lastName.alltime))

# Expected output

[CCC]==>listcomp: 0.00002, 0.00002
[CCC]==>listcomp: 1.19636, 1.19638
[0, 2, 4, 6, 8]
allTime = 1.19637775192
[MMM]==>mapcall: 0.00002, 0.00002
[MMM]==>mapcall: 2.29260, 2.29262
[0, 2, 4, 6, 8]
allTime = 2.2926232943

giveRaise: 0.00001, 0.00001
giveRaise: 0.00001, 0.00002
55000.0 120000.0
**lastName: 0.00001, 0.00001
**lastName: 0.00001, 0.00002
Smith Jones
0.00002 0.00002
```

2.□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□□□(-0)□□□□□□□□□□□□□□□□□□□

```
traceMe = False
def trace(*args):
    if traceMe: print('[' + ' '.join(map(str, args)) + ']')

def accessControl(failIf):
    def onDecorator(aClass):
        if not __debug__:
            return aClass
        else:
            class onInstance:
```

```
                def __init__(self, *args, **kargs):
                    self._wrapped = aClass(*args, **kargs)
                def __getattr__(self, attr):
                    trace('get:', attr)
                    if failIf(attr):
                        raise TypeError('private attribute fetch: ' + attr)
                    else:
                        return getattr(self._wrapped, attr)
                def __setattr__(self, attr, value):
                    trace('set:', attr, value)
                    if attr == '_onInstance__wrapped':
                        self.__dict__[attr] = value
                    elif failIf(attr):
                        raise TypeError('private attribute change: ' + attr)
                    else:
                        setattr(self._wrapped, attr, value)
            return onInstance
    return onDecorator

def Private(*attributes):
    return accessControl(failIf=(lambda attr: attr in attributes))

def Public(*attributes):
    return accessControl(failIf=(lambda attr: attr not in attributes))

# Test code: split me off to another file to reuse decorator

@Private('age')                          # Person = Private('age')(Person)
class Person:                            # Person = onInstance with state
    def __init__(self, name, age):
        self.name = name
        self.age = age                   # Inside accesses run normally

X = Person('Bob', 40)
print(X.name)                            # Outside accesses validated
X.name = 'Sue'
print(X.name)
#print(X.age)           # FAILS unles "python -O"
#X.age = 999            # ditto
#print(X.age)           # ditto

@Public('name')
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

X = Person('bob', 40)                    # X is an onInstance
print(X.name)                            # onInstance embeds Person
X.name = 'Sue'
print(X.name)
#print(X.age)        # FAILS unless "python -O main.py"
#X.age = 999         # ditto
#print(X.age)        # ditto
```

□39□　□□

□□□

□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□Python□□□□Tim Peters□comp.lang.python□□□□□□□□□□Tim Peters□□□□□Python□□□□"import this"□□□□□□

[□□]□99%□□□□□□□□□□□□□□□□□□□

□□□

□□□__getattr__□__setattr__□__getattribute__□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□class□□□□□□□□□□□□□□□□□□□

“□□”□□□□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
def extra(self, arg): ...

class Client1: ...                          # Client augments: too distributed
if required():
    Client1.extra = extra

class Client2: ...
if required():
    Client2.extra = extra

class Client3: ...
if required():
    Client3.extra = extra

X = Client1()
X.extra()
```

□□□□□□□□class□□□□□□□□□□□□□□□□□□□

```
def extra(self, arg): ...

def extras(Class):                          # Manager function: too manual
    if required():
        Class.extra = extra

class Client1: ...
extras(Client1)

class Client2: ...
extras(Client2)

class Client3: ...
extras(Client3)

X = Client1()
X.extra()
```

□□□

```
def extra(self, arg): ...

class Extras(type):
    def __init__(Class, classname, superclasses, attributedict):
        if required():
            Class.extra = extra

class Client1(metaclass=Extras): ...    # Metaclass declaration only
class Client2(metaclass=Extras): ...    # Client class is instance of meta
class Client3(metaclass=Extras): ...

X = Client1()                           # X is instance of Client1
X.extra()
```

□□□□□□□□□□□□Python□class□□□□□□□□□□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□class□□□□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□type□□□□□

□□□□□□□/□□□□□□□

```
C:\misc> c:\python30\python
>>> class C: pass              # 3.0 class object (new-style)
...
>>> X = C()                    # Class instance object

>>> type(X)                    # Instance is instance of class
<class '__main__.C'>
>>> X.__class__                # Instance's class
<class '__main__.C'>

>>> type(C)                    # Class is instance of type
<class 'type'>
>>> C.__class__                # Class's class is type
<class 'type'>
```

□□□□□□□□□□□□——□□type□□□□□□□□□□□□

```
C:\misc> c:\python26\python
>>> class C(object): pass                    # In 2.6 new-style classes,
...                                          # classes have a class too
>>> X = C()

>>> type(X)
<class '__main__.C'>
>>> type(C)
<type 'type'>

>>> X.__class__
<class '__main__.C'>
>>> C.__class__
<type 'type'>
```

□□□Python 2.6□□□□□□□□□□——□□□□□□□□Python□□□□□□□□□□□□□□□□__class__□□□□□□Python 2.6□□□□□□□□□□□□□□□type□□□□□□□□□□□□□□□

```
C:\misc> c:\python26\python
>>> class C: pass                            # In 2.6 classic classes,
...                                          # classes have no class themselves
>>> X = C()

>>> type(X)
<type 'instance'>
>>> type(C)
<type 'classobj'>

>>> X.__class__
<class __main__.C at 0x005F85A0>
>>> C.__class__
AttributeError: class C has no attribute '__class__'
```

□□□Type□□□

□□□□□□□□□□□□□Python 3.0□□□□type□□□□□□□□□□□□□

Class□□□□

□□□type□□□□

```
class Spam(Eggs):                # Inherits from Eggs
    data = 1                     # Class data attribute
    def meth(self, arg):         # Class method attribute
        pass
```

Python□□□

□□□□

□□□□□□□□□□□□□□□□type□□□□□□□□□Python□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□

__new__和__init__□□□

□□□□□□□□

□□□□□□□__init__□□□□type□□□__call__□□□□□□__new__□□□□□□□□□□__init__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class MetaOne(type):
    def __new__(meta, classname, supers, classdict):
        print('In MetaOne.new: ', classname, supers, classdict, sep='\n...')
        return type.__new__(meta, classname, supers, classdict)

    def __init__(Class, classname, supers, classdict):
        print('In MetaOne init:', classname, supers, classdict, sep='\n...')
        print('...init class object:', list(Class.__dict__.keys()))

class Eggs:
    pass

print('making class')
class Spam(Eggs, metaclass=MetaOne):    # Inherits from Eggs, instance of Meta
    data = 1                            # Class data attribute
    def meth(self, arg):                # Class method attribute
        pass

print('making instance')


 X = Spam()
 print('data:', X.data)
```

□□□□□□□□□□□□□□□□□□□

```
making class
In MetaOne.new:
...Spam
...(<class '__main__.Eggs'>,)
...{'__module__': '__main__', 'data': 1, 'meth': <function meth at 0x02AAB810>}
In MetaOne init:
...Spam
...(<class '__main__.Eggs'>,)
...{'__module__': '__main__', 'data': 1, 'meth': <function meth at 0x02AAB810>}
...init class object: ['__module__', 'data', 'meth', '__doc__']
making instance
data: 1
```

□□□□□□□□

□□□□□□type□□□__new__□__init__□□□□□□□

```
making class
In MetaFunc:
...Spam
...(<class '__main__.Eggs'>,)
...{'__module__': '__main__', 'data': 1, 'meth': <function meth at 0x02B8B6A8>}
making instance
data: 1
```

□□□□□□□□□□

□□□□□□□□□□OOP□□□□□□□□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□□type□□□__call__□□□□□□□□□□□□□□□

```
# __call__ can be redefined, metas can have metas

class SuperMeta(type):
    def __call__(meta, classname, supers, classdict):
        print('In SuperMeta.call: ', classname, supers, classdict, sep='\n...')
        return type.__call__(meta, classname, supers, classdict)

class SubMeta(type, metaclass=SuperMeta):
    def __new__(meta, classname, supers, classdict):
        print('In SubMeta.new: ', classname, supers, classdict, sep='\n...')
        return type.__new__(meta, classname, supers, classdict)

    def __init__(Class, classname, supers, classdict):
        print('In SubMeta init:', classname, supers, classdict, sep='\n...')
        print('...init class object:', list(Class.__dict__.keys()))

class Eggs:
    pass

print('making class')
class Spam(Eggs, metaclass=SubMeta):
    data = 1
    def meth(self, arg):
        pass

print('making instance')
X = Spam()
print('data:', X.data)
```

□□□□□□□□□□□□□□3□□□□□□□□□□□□□□□□□□□□□□□□□□□type□□□□□□□□□□

```
making class
In SuperMeta.call:
...Spam
...(<class '__main__.Eggs'>,)
...{'__module__': '__main__', 'data': 1, 'meth': <function meth at 0x02B7BA98>}


In SubMeta.new:
...Spam
...(<class '__main__.Eggs'>,)
...{'__module__': '__main__', 'data': 1, 'meth': <function meth at 0x02B7BA98>}
In SubMeta init:
...Spam
...(<class '__main__.Eggs'>,)
...{'__module__': '__main__', 'data': 1, 'meth': <function meth at 0x02B7BA98>}
...init class object: ['__module__', 'data', 'meth', '__doc__']
making instance
data: 1
```

□□□□□□□□□□□□

□□□□□□□□□□□

```
class SuperMeta:
    def __call__(self, classname, supers, classdict):
        print('In SuperMeta.call: ', classname, supers, classdict, sep='\n...')
        Class = self.__New__(classname, supers, classdict)
        self.__Init__(Class, classname, supers, classdict)
        return Class

class SubMeta(SuperMeta):
    def __New__(self, classname, supers, classdict):
        print('In SubMeta.new: ', classname, supers, classdict, sep='\n...')
        return type(classname, supers, classdict)

    def __Init__(self, Class, classname, supers, classdict):
        print('In SubMeta init:', classname, supers, classdict, sep='\n...')
        print('...init class object:', list(Class.__dict__.keys()))

class Eggs:
    pass

print('making class')
class Spam(Eggs, metaclass=SubMeta()):          # Meta is normal class instance
    data = 1                                    # Called at end of statement
    def meth(self, arg):
        pass

print('making instance')
X = Spam()
print('data:', X.data)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□type□□□__new__□__init__□□□□□□□□□□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

```
class MetaOne(type):
    def __new__(meta, classname, supers, classdict):          # Redefine type method
        print('In MetaOne.new:', classname)
        return type.__new__(meta, classname, supers, classdict)
    def toast(self):
        print('toast')

class Super(metaclass=MetaOne):          # Metaclass inherited by subs too
    def spam(self):                      # MetaOne run twice for two classes
        print('spam')

class C(Super):                          # Superclass: inheritance versus instance
    def eggs(self):                      # Classes inherit from superclasses
        print('eggs')                    # But not from metclasses

X = C()
X.eggs()              # Inherited from C
X.spam()              # Inherited from Super
X.toast()             # Not inherited from metaclass
```

□□□

□□□□□□□□□□

□□□□□□□□□

```
# Extend manually - adding new methods to classes

class Client1:
    def __init__(self, value):
```

```
        self.value = value
    def spam(self):
        return self.value * 2

class Client2:
    value = 'ni?'

def eggsfunc(obj):
    return obj.value * 4

def hamfunc(obj, value):
    return value + 'ham'

Client1.eggs = eggsfunc
Client1.ham = hamfunc

Client2.eggs = eggsfunc
Client2.ham = hamfunc

X = Client1('Ni!')
print(X.spam())
print(X.eggs())
print(X.ham('bacon'))

Y = Client2()
print(Y.eggs())
print(Y.ham('bacon'))
```

□□□

□□□

□□□□□□□

□□□□□□

```
# Extend with a metaclass - supports future changes better

def eggsfunc(obj):
    return obj.value * 4
def hamfunc(obj, value):
    return value + 'ham'

class Extender(type):
    def __new__(meta, classname, supers, classdict):
        classdict['eggs'] = eggsfunc
        classdict['ham'] = hamfunc
        return type.__new__(meta, classname, supers, classdict)

class Client1(metaclass=Extender):
    def __init__(self, value):
        self.value = value
    def spam(self):
        return self.value * 2

class Client2(metaclass=Extender):
    value = 'ni?'

X = Client1('Ni!')
print(X.spam())
print(X.eggs())
print(X.ham('bacon'))

Y = Client2()
print(Y.eggs())
print(Y.ham('bacon'))
```

□□□

```
Ni!Ni!
Ni!Ni!Ni!Ni!
baconham
ni?ni?ni?ni?
baconham
```

□□□

□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□

```
# Extend with a decorator: same as providing __init__ in a metaclass

def eggsfunc(obj):
    return obj.value * 4

def hamfunc(obj, value):
    return value + 'ham'

def Extender(aClass):
    aClass.eggs = eggsfunc              # Manages class, not instance
    aClass.ham = hamfunc                # Equiv to metaclass __init__
    return aClass


@Extender
class Client1:                          # Client1 = Extender(Client1)
def __init__(self, value):              # Rebound at end of class stmt
    self.value = value
def spam(self):
    return self.value * 2

@Extender
class Client2:
    value = 'ni?'

X = Client1('Ni!')                      # X is a Client1 instance
print(X.spam())
print(X.eggs())
print(X.ham('bacon'))

Y = Client2()
print(Y.eggs())
print(Y.ham('bacon'))
```

□□□

□□□

```
Trace: name
Bob
Trace: pay
2000
```

□□□

□□□□□□class□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□

```
            values = (label, func.__name__, elapsed, onCall.alltime)
            print(format % values)
            return result
        onCall.alltime = 0
        return onCall
    return onDecorator
```

□□□

```
call 1 to __init__
call 2 to __init__
Bob Smith Sue Jones
call 1 to giveRaise
110000.0
call 1 to lastName
call 2 to lastName
Smith Jones
```

□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□□□

□□□

```
# If using timer: total time per method

print('-'*40)
print('%.5f' % Person.__init__.alltime)
print('%.5f' % Person.giveRaise.alltime)
print('%.5f' % Person.lastName.alltime)
```

□□□□□□□□——□□□□□□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

□□□

```
@decorateAll(tracer)                    # Decorate all with tracer
@decorateAll(timer())                   # Decorate all with timer, defaults
@decorateAll(timer(label='@@'))         # Same but pass a decorator argument
```

□□□

```
@@__init__: 0.00001, 0.00001
@@__init__: 0.00001, 0.00002
Bob Smith Sue Jones
@@giveRaise: 0.00001, 0.00001
110000.0
@@lastName: 0.00001, 0.00001
@@lastName: 0.00001, 0.00002
Smith Jones
----------------------------------------
0.00002
0.00001
0.00002
```

□□□

□□□——□□□□□□□□□□□□□□□□□□□

□□□

□□□□

□□□

□□”

□□□□

1.□□□□□□

2.□□□□□□□□□□□

3.□□□□□□□□□□□□□□□□□□□□□□

4.□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□type□□□□□□□□□□□□type□□□□□□□□□□□□□□□□□□□

□□□□ □□

□□A □□□□□□

□□□□□

·在Mac OS X中，打开Terminal（在(Applications→Utilities→Terminal)）中，输入Python。

·在Linux或Unix中，在shell中，输入python命令，或者找到python安装的目录，并输入"python"（/usr/bin或/usr/local/bin）。

在学习Python之前，有一个问题困扰着很多人，Python有两个版本，Python 3.0和Python 2.6，两个版本之间有很大的差别，不能兼容，那么我们学习哪个版本呢？

其实，我们学习Python，学习的是Python 2.X，因为现在的Python 3.0，并没有得到广泛的应用，Python 2.6是过渡版本，Python 2.5以及之前的版本，官方已经不再支持，所以，我们学习Python，从

□□□□□Python

□□□□□Python□□□□□□□□□□□□□□□Python□□□□□□□□Web□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□Python□□□□□http://www.python.org□□□□□□□□□□Python□□□□□□□□□Downloads□□□□□□□□□□□□□□□□□□□□□□□□□□□□Windows□□□□□□□□□□□□□□□□□□□□□□□□Mac OS X□□□□□□□□□□□□□□□□□□□□□□□Linux□Unix□OS X□□□□□□□□□□□□□□□□□□□

□□□□Python□Linux□□□□□□□□□□□□Web□□□□□Linux□RPM□□rpm□□□□□□Python□Web□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Google Web□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□Linux CD□□□□□□□Python□□□□□□□

□□□□

□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□

Windows

□Windows□□Python□□□□□□MSI□□□□□□□□□□□□□□□□□□□

□□□□□□□□

Linux

□Linux□□Python□□□□□□□□□RPM□□□□□□□□□

Vista□□□□□□□□□□□□□□□□□□□□MSI□□□□□□□Python□□□□□□C□□□□□□□□□□□□C:\PythonXX□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□Vista□□□□□□□□□□□□□□□MSI□□□□□□□□□□□□□□□□□□□

□□Python

□□□Python□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□

PATH□□□

□□□□□tkinter GUI□□□□□Python 2.6□□Tkinter□□□□□□□A-1□□□GUI□□□□□Tcl□Tk□□□□□□□□□□□□□□□PYTHONPATH□□□□□□□□□□Windows□□□□□□□□tkinter□□Python□□□□□□□□Tcl□Tk□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□.pth□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□Python□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□

□□Linux□tkinter□□IDLE□GUI□□□

□2□□□□□IDLE□□□Python tkinter GUI□□□tkinter□GUI□□□□□□□Windows□□□□□□Python□□□□□□□□□□□□□Linux□□□□□□GUI□□□□□□□□□□□□□□□Linux□□Python□□GUI□□□□□□□□□□yumt kinter□□□□□□□tkinter□□□□□□□□□□□□□□□□□□yum□□□□□Linux□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□Python□□□□□□□

□□shell□□□□□□□□□□□□□□□□□

DOS□□□Windows□

□□□□□□MS-DOS□□□□Windows□□□□□□C:\autoexec.bat□□□□□□□□□□□□□□□□□

□□□□□□□□□Windows□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□”→“□□”□□□□□regedit□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□

Python□□□□□

□□□□□□□□□□□□□□Python□□□□□shell□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□-c□□□□□□□□□□□□□□□□□□-□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
c:\Python30> python -c "print(2 ** 100)"          # Read code from command argument
1267650600228229401496703205376

c:\Python30> python -c "import main"              # Import a file to run its code
['-c']

c:\Python30> python - < main.py a b -c            # Read code from standard input
['-', 'a', 'b', '-c']

c:\Python30> python - a b -c < main.py            # Same effect as prior line
['-', 'a', 'b', '-c']
```

-m□□□□□Python□□□□□□□□sys.path□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□__main__□□□□□□□□".py"□□□□□□□□□□□□□□□□

```
c:\Python30> python -m main a b -c                # Locate/run module as script
['c:\\Python30\\main.py', 'a', 'b', '-c']
```

-m□□□□□□□□□□□□□□□□□□□

```
(Pdb) c
c:\Python30> python -m profile main.py a b -c          # Profile a script
c:\Python30> python -m cProfile main.py a b -c         # Low-overhead profiler
```

□□□"python"□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□-O□□□□□□□□Python□-u□□□□□□unbuffered□□-i□□□□□□□□□□□□□□□□

```
c:\Python30> python -u main.py a b -c                  # Unbuffered output streams
```

Python 2.6□□□□□□□□□□□□□□Python 3.0□□□□□□-3,-0□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□-t□□□□12□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□

Python□□□□□

□□B □□□□□□□□□

□□□□ □□□□

□□□3□“□□□□ □□□”□□□□□

1.□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□IDLE□shell□□□□□□□□

```
% python
...copyright information lines...
>>> "Hello World!"
'Hello World!'
>>>                # Use Ctrl-D or Ctrl-Z to exit, or close window
```

2.□□□□□□□□□□□□□□□□□□module1.py□□□□□shell□□□□□□□□□□□□

```
print('Hello module world!')

% python module1.py
Hello module world!
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□IDLE□Run/Run Module□□□□□

3.□□□

□□□□□□try□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□L□□□□□□□□□□

□□□□ □□□□□

□□□9□“□□□□ □□□”□□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“;”□□□□□□□□□□□□□□□□□□□□“;”□□□□□□□□□□□□□□□□□□□

```
# Numbers

>>> 2 ** 16                                  # 2 raised to the power 16
65536
>>> 2 / 5, 2 / 5.0                           # Integer / truncates in 2.6, but not 3.0
(0.40000000000000002, 0.40000000000000002)

# Strings

>>> "spam" + "eggs"                          # Concatenation
'spameggs'
>>> S = "ham"
>>> "eggs " + S
'eggs ham'
>>> S * 5                                    # Repetition
'hamhamhamhamham'
>>> S[:0]                                    # An empty slice at the front -- [0:0]
''                                           # Empty of same type as object sliced

>>> "green %s and %s" % ("eggs", S)          # Formatting
'green eggs and ham'
>>> 'green {0} and {1}'.format('eggs', S)
'green eggs and ham'

# Tuples

>>> ('x',)[0]                                # Indexing a single-item tuple
'x'
>>> ('x', 'y')[1]                            # Indexing a 2-item tuple
'y'

# Lists

>>> L = [1,2,3] + [4,5,6]                    # List operations
>>> L, L[:], L[:0], L[-2], L[-2:]
([1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6], [], 5, [5, 6])
>>> ([1,2,3]+[4,5,6])[2:4]
[3, 4]
>>> [L[2], L[3]]                             # Fetch from offsets; store in a list
[3, 4]
>>> L.reverse(); L                           # Method: reverse list in-place
[6, 5, 4, 3, 2, 1]
>>> L.sort(); L                              # Method: sort list in-place
[1, 2, 3, 4, 5, 6]
>>> L.index(4)                               # Method: offset of first 4 (search)
3

# Dictionaries

>>> {'a':1, 'b':2}['b']                      # Index a dictionary by key
2
>>> D = {'x':1, 'y':2, 'z':3}
>>> D['w'] = 0                               # Create a new entry
>>> D['x'] + D['w']
1
>>> D[(1,2,3)] = 4                           # A tuple used as a key (immutable)
```

```
>>> D
{'w': 0, 'z': 3, 'y': 2, (1, 2, 3): 4, 'x': 1}

>>> list(D.keys()), list(D.values()), (1,2,3) in D          # Methods, key test
(['w', 'z', 'y', (1, 2, 3), 'x'], [0, 3, 2, 4, 1], True)

# Empties

>>> [[]], ["",[],(),{},None]                    # Lots of nothings: empty objects
([[]], ['', [], (), {}, None])
```

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□L[4]□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

```
>>> L = [1, 2, 3, 4]
>>> L[4]
Traceback (innermost last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
>>> L[-1000:100]
[1, 2, 3, 4]
>>> L[3:1]
[]
>>> L
[1, 2, 3, 4]
>>> L[3:1] = ['?']
>>> L
[1, 2, 3, '?', 4]
```

3.□□□□□□□□□□□□del□□□□□□□□□□□□

```
[1, 2, 4]
>>> del L[0]
>>> L
[2, 4]
>>> del L[1:]
>>> L
[2]
>>> L[1:2] = 1
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: illegal argument type for built-in operation
```

4.□□□□□□□□□□X□Y□□□□□□□□□□□□□□□□=□□□□□□□□Python□□□□□□□□□□□□□□□□

```
>>> D = {}
>>> D[1] = 'a'
>>> D[2] = 'b'
>>> D[(1, 2, 3)] = 'c'
>>> D
{1: 'a', 2: 'b', (1, 2, 3): 'c'}
```

6.□□□□□□□□□□□□□□□□□□□□□□□□D['d']□□□□□□□□□□□□□□□□□□□

7.□□□□□□□□□□

·+□□□□□□□□□/□□□□□□□□□□□+□□□□+□□□□

·+□□□□□□□□□□□□□□□

·append□□□□□□□□□□□□□□□□□□□□□□□□□□□□□append□□

·□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> "x" + 1
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: illegal argument type for built-in operation
>>>
>>> {} + {}
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: bad operand type(s) for +
>>>
>>> [].append(9)
>>> "".append('s')
Traceback (innermost last):
  File "<stdin>", line 1, in ?
AttributeError: attribute-less object
>>>
>>> list({}.keys())                          # list needed in 3.0, not 2.6
[]
>>> [].keys()
Traceback (innermost last):
  File "<stdin>", line 1, in ?
AttributeError: keys
>>>
>>> [][:]
[]
>>> ""[:]
''
```

8.□□□

```
>>> me = {'name':('John', 'Q', 'Doe'), 'age':'?', 'job':'engineer'}
>>> me['job']
'engineer'
>>> me['name'][2]
'Doe'
```

11.□□□□□□□Python□□□□□□□□□□□□□□□ls□UNIX□□□□Windows□□□□dir□□

```
# File: maker.py
file = open('myfile.txt', 'w')
file.write('Hello file world!\n')          # Or: open().write()
file.close()                               # close not always needed

# File: reader.py
file = open('myfile.txt')                  # 'r' is default open mode
print(file.read())                         # Or print(open().read())
```

```
% python maker.py
% python reader.py
Hello file world!
% ls -l myfile.txt
-rwxrwxrwa  1 0       0 19 Apr 13 16:33 myfile.txt
```

□□□□ □□□□□

□□□15□“□□□□ □□□”□□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> S = 'spam'
>>> for c in S:
...     print(ord(c))
...
115
112
97
109

>>> x = 0
>>> for c in S: x += ord(c)                # Or: x = x + ord(c)
...
>>> x
433

>>> x = []
>>> for c in S: x.append(ord(c))
...
>>> x
[115, 112, 97, 109]

>>> list(map(ord, S))                      # list() required in 3.0, not 2.6
[115, 112, 97, 109]
```

2.□□□□□□□□□□□□□□□□□□□□□□\a□50□□□□□□□□□□□□□□□□□□IDLE□□□□□□□□□□□□□□□□□□□

3.□□□

```
# a

L = [1, 2, 4, 8, 16, 32, 64]
X = 5

i = 0
while i < len(L):
    if 2 ** X == L[i]:
        print('at index', i)
        break
    i += 1
else:
    print(X, 'not found')

# b

L = [1, 2, 4, 8, 16, 32, 64]
X = 5

for p in L:
    if (2 ** X) == p:
        print((2 ** X), 'was found at', L.index(p))
        break
else:
    print(X, 'not found')

# c

L = [1, 2, 4, 8, 16, 32, 64]
X = 5

if (2 ** X) in L:
    print((2 ** X), 'was found at', L.index(2 ** X))
else:
    print(X, 'not found')

# d

X = 5
L = []
for i in range(7): L.append(2 ** i)
```

```
print(L)

if (2 ** X) in L:
    print((2 ** X), 'was found at', L.index(2 ** X))
else:
    print(X, 'not found')

# f

X = 5
L = list(map(lambda x: 2**x, range(7)))          # or [2**x for x in range(7)]
print(L)                                         # list() to print all in 3.0, not 2.6

if (2 ** X) in L:
    print((2 ** X), 'was found at', L.index(2 ** X))
else:
    print(X, 'not found')
```

□□□□ □□

□□□20□“□□□□ □□□”□□□□

1.□□□□□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□□□□

```
def adder(x, y):
    return x + y

print(adder(2, 3))
print(adder('spam', 'eggs'))
print(adder(['a', 'b'], ['c', 'd']))

% python mod.py
5
spameggs
['a', 'b', 'c', 'd']
```

3.□□□□□□□□□adders.py□□□□□□□□□adder□□□□□□□□□

```
def adder1(*args):
    print('adder1', end=' ')
    if type(args[0]) == type(0):                    # Integer?
         sum = 0                                    # Init to zero
    else:                                           # else sequence:
         sum = args[0][:0]                          # Use empty slice of arg1
    for arg in args:
        sum = sum + arg
    return sum

def adder2(*args):
    print('adder2', end=' ')
    sum = args[0]                                   # Init to arg1
    for next in args[1:]:
        sum += next                                 # Add items 2..N
    return sum

for func in (adder1, adder2):
    print(func(2, 3, 4))
    print(func('spam', 'eggs', 'toast'))
    print(func(['a', 'b'], ['c', 'd'], ['e', 'f']))
```

```
% python adders.py
adder1 9
adder1 spameggstoast
adder1 ['a', 'b', 'c', 'd', 'e', 'f']
adder2 9
adder2 spameggstoast
adder2 ['a', 'b', 'c', 'd', 'e', 'f']
```

4.□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□mod.py□□□□□□□□□□□□□□□□□□□□□□□**args□□□□□□□□□□[□□□for x in args.keys():use args[x]]□□□□□args.values()□□□□□□□□□*args□□□□□□□□

```
def adder(good=1, bad=2, ugly=3):
    return good + bad + ugly

print(adder())
print(adder(5))
print(adder(5, 6))
```

```
print(adder(5, 6, 7))
print(adder(ugly=7, good=6, bad=5))

% python mod.py
6
10
14
18
18

# Second part solutions

def adder1(*args):                          # Sum any number of positional args
    tot = args[0]
    for arg in args[1:]:
        tot += arg
    return tot

def adder2(**args):                         # Sum any number of keyword args
    argskeys = list(args.keys())            # list needed in 3.0!
    tot = args[argskeys[0]]
    for key in argskeys[1:]:
        tot += args[key]
    return tot

def adder3(**args):                         # Same, but convert to list of values
    args = list(args.values())              # list needed to index in 3.0!
    tot = args[0]
    for arg in args[1:]:
        tot += arg
    return tot

def adder4(**args):                         # Same, but reuse positional version
    return adder1(*args.values())

print(adder1(1, 2, 3),       adder1('aa', 'bb', 'cc'))
print(adder2(a=1, b=2, c=3), adder2(a='aa', b='bb', c='cc'))
print(adder3(a=1, b=2, c=3), adder3(a='aa', b='bb', c='cc'))
print(adder4(a=1, b=2, c=3), adder4(a='aa', b='bb', c='cc'))
```

5.□□6.□□□□□□□□5□6□□□□□□dicts.py□□□□□□□□

□□□□□□□□X[:]□□□□□□□□□□□□□□□□□□□□□□8□□□□□□□□□□□□□□□□□□□□□□□□□□□□e=d□□□□□□□□□□□□□□□□□□□□□□□□□□□□d□□□□□□e□

```
def copyDict(old):
    new = {}
    for key in old.keys():
        new[key] = old[key]
    return new

def addDict(d1, d2):
```

```
    new = {}
    for key in d1.keys():
        new[key] = d1[key]
    for key in d2.keys():
        new[key] = d2[key]
    return new
```

```
% python
>>> from dicts import *
>>> d = {1: 1, 2: 2}
>>> e = copyDict(d)
>>> d[2] = '?'
>>> d
{1: 1, 2: '?'}
>>> e
{1: 1, 2: 2}

>>> x = {1: 1}
>>> y = {2: 2}
>>> z = addDict(x, y)
>>> z
{1: 1, 2: 2}
```

6.□□5□

7.□□□

```
def f1(a, b): print(a, b)                # Normal args
def f2(a, *b): print(a, b)               # Positional varargs
def f3(a, **b): print(a, b)              # Keyword varargs
def f4(a, *b, **c): print(a, b, c)       # Mixed modes
def f5(a, b=2, c=3): print(a, b, c)      # Defaults
def f6(a, b=2, *c): print(a, b, c)       # Defaults and positional varargs

% python
>>> f1(1, 2)                           # Matched by position (order matters)
1 2
>>> f1(b=2, a=1)                       # Matched by name (order doesn't matter)
1 2

>>> f2(1, 2, 3)                        # Extra positionals collected in a tuple
1 (2, 3)

>>> f3(1, x=2, y=3)                    # Extra keywords collected in a dictionary
1 {'x': 2, 'y': 3}

>>> f4(1, 2, 3, x=2, y=3)              # Extra of both kinds
1 (2, 3) {'x': 2, 'y': 3}

>>> f5(1)                              # Both defaults kick in
1 2 3

>>> f5(1, 4)                           # Only one default used
1 4 3

>>> f6(1)                              # One argument: matches "a"
1 2 ()

>>> f6(1, 3, 4)                        # Extra positional collected
1 3 (4,)
```

8.□□□

```
#from __future__ import division

def prime(y):
    if y <= 1:                                  # For some y > 1
        print(y, 'not prime')
    else:
        x = y // 2                              # 3.0 / fails
        while x > 1:
            if y % x == 0:                      # No remainder?
                print(y, 'has factor', x)
                break                           # Skip else
            x -= 1
        else:
            print(y, 'is prime')

prime(13); prime(13.0)
prime(15); prime(15.0)
prime(3); prime(2)
prime(1); prime(-3)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□//□□□□□□□□□□□□□

```
% python primes.py
13 is prime
13.0 is prime
15 has factor 5
15.0 has factor 5.0
3 is prime
2 is prime
1 not prime
-3 not prime
```

□□□□□□□□□□□□□□□□□□□

□□□range(y,1,-1)□□□□□while□□□time□□□□□□□□□□□□□□□□□timer□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
def timer(reps, func, *args):
    import time
    start = time.clock()
    for i in range(reps):
        func(*args)
    return time.clock() - start
```

9.□□□□□□□□□□□□□□□□□□□

10.□□□□□□□□□□□□□□□3□□□□□□□□□□□□□□□□□Python 2.6□Python 3.0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□3□□□□□□□□□□□□

```
# File mytimer.py (2.6 and 3.0)

...same as listed in Chapter 20...

# File timesqrt.py

import sys, mytimer
reps = 10000
repslist = range(reps)                    # Pull out range list time for 2.6

from math import sqrt                     # Not math.sqrt: adds attr fetch time
def mathMod():
    for i in repslist:
        res = sqrt(i)
    return res

def powCall():
    for i in repslist:
        res = pow(i, .5)
    return res

def powExpr():


 for i in repslist:
     res = i ** .5
 return res

print(sys.version)
for tester in (mytimer.timer, mytimer.best):
    print('<%s>' % tester.__name__)
    for test in (mathMod, powCall, powExpr):
        elapsed, result = tester(test)
        print ('-'*35)
        print ('%s: %.5f => %s' %
               (test.__name__, elapsed, result))
```

□□□□□Python 3.0□Python 2.6□□□□□□□□□□□□□□□□□□□math□□□**□□□□□□□**□□□□pow□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□

□□□□□□□□□□□□□□Python 3.0□□□Python 2.6□□□□Python 3.1□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
c:\misc> c:\python30\python timesqrt.py
3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)]
<timer>
-----------------------------------
mathMod: 5.33906 => 99.994999875
-----------------------------------
powCall: 7.29689 => 99.994999875
-----------------------------------
powExpr: 5.95770 => 99.994999875
<best>
-----------------------------------
mathMod: 0.00497 => 99.994999875
-----------------------------------
powCall: 0.00671 => 99.994999875
-----------------------------------
powExpr: 0.00540 => 99.994999875


c:\misc> c:\python26\python timesqrt.py
2.6.1 (r261:67517, Dec 4 2008, 16:51:00) [MSC v.1500 32 bit (Intel)]
<timer>
-----------------------------------
mathMod: 2.61226 => 99.994999875
-----------------------------------
powCall: 4.33705 => 99.994999875
-----------------------------------
powExpr: 3.12502 => 99.994999875
<best>
-----------------------------------
mathMod: 0.00236 => 99.994999875
-----------------------------------
powCall: 0.00402 => 99.994999875
-----------------------------------
powExpr: 0.00287 => 99.994999875
```

□□□Python 3.0□□□□□□□□for□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□50□□□□□□□1 000 000□□□□□□□□□□

□□□□ □□

□□□24□“□□□□ □□□”□□□□

1.□□□□

```
def countLines(name):
    tot = 0
    for line in open(name): tot += 1
    return tot

def countChars(name):
    tot = 0
    for line in open(name): tot += len(line)
    return tot
```

□UNIX□□□□□□□wc□□□□□□□□Windows□□□□□□□□

```
    return len(file.read())

def test(name):
    file = open(name)                                   # Pass file object
    return countLines(file), countChars(file)           # Open file only once
```

```
>>> import mymod2
>>> mymod2.test("mymod2.py")
(11, 392)
```

2.from/from*□□□□from*□□□□*□□countChars□□□□□□□□

```
% python
>>> from mymod import *
>>> countChars("mymod.py")
291
```

3.__main__□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
def countLines(name):
    file = open(name)
    return len(file.readlines())

def countChars(name):
    return len(open(name).read())

def test(name):                                         # Or pass file object
    return countLines(name), countChars(name)           # Or return a dictionary

if __name__ == '__main__':
    print(test('mymod.py'))
```

```
% python mymod.py
(13, 346)
```

□□□

□□myclient□□□import□□□from□□□□□□□□□□□□myclient□□□mymod□□□□□

```
import myclient
myclient.mymod.countLines(...)

from myclient import mymod
mymod.countChars(...)
```

□□□□□□□□□□□□

□□□□□□□□□□mypkg□□□□□□□__init__.py□mymod.py□mypkg□□□□□□__init__.py□□□□□□□□□□□□mypkg□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□□□□□□□□□

```
C:\python30> mkdir mypkg
C:\Python30> move mymod.py mypkg\mymod.py
C:\Python30> edit mypkg\__init__.py


...coded a print statement...
C:\Python30> python
>>> import mypkg.mymod
initializing mypkg
>>> mypkg.mymod.countLines('mypkg\mymod.py')
13
>>> from mypkg.mymod import countChars
>>> countChars('mypkg\mymod.py')
346
```

6.□□□□□□□□□□□□□□□□□□□□□□□changer.py□□□□□□□□□□□□□□□□□

7.□□□□□□□□□□□□□□□□□□□□□recur2□□□□□□□□□□□□□recur1□□import□□□□recur2□from□

□□□□□□□□□□□□□□recur2□□□□□□recur1□recur2□□□□□□□□□□□□recur2□□□□□□□□□□□□□□□□□recur1□□□□recur2□□□□□□□□□□□□import□□□from□□□□□□□Python□□□□□□□□□□□□recur2□□□□□□□□□□□□□recur1□□□□□□□□□□□□□□□□recur2□□□□□□□□□□□□□□□□from□□recur1□□□□□□□□□□□□□□Y□□□□□□□□□□□□□□□□□□□

□□□□ □□OOP

□□□31□“□□□□ □□□”□□□□

1.□□□□□□□□□□□□□□□□□adder.py□□□□□□□□□□□□□□□□□□□__add__□□□□□□□□□□□□□□□□□□□

```
        new = {}
        for k in x.keys(): new[k] = x[k]
        for k in y.keys(): new[k] = y[k]
        return new

% python
>>> from adder import *
>>> x = Adder()
>>> x.add(1, 2)
not implemented!
>>> x = ListAdder()
>>> x.add([1], [2])
[1, 2]
>>> x = DictAdder()
>>> x.add({1:1}, {2:2})
{1: 1, 2: 2}

>>> x = Adder([1])
>>> x + [2]
not implemented!
>>>
>>> x = ListAdder([1])
>>> x + [2]
[1, 2]
>>> [2] + x
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: __add__ nor __radd__ defined for these operands
```

□□□□□□□□□□□□□□□□□□□□+□□□□□□□□□□□□□□□□□□□□□__radd__□□□□□29□□□□□□□□

□□□□□□□□□□□□□□□□□□□□add□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class Adder:
    def __init__(self, start=[]):
        self.data = start
    def __add__(self, other):              # Pass a single argument
        return self.add(other)             # The left side is in self
    def add(self, y):
        print('not implemented!')

class ListAdder(Adder):
    def add(self, y):
        return self.data + y

class DictAdder(Adder):
    def add(self, y):
        pass                               # Change to use self.data instead of x

x = ListAdder([1, 2, 3])
y = x + [4, 5, 6]
print(y)                                   # Prints [1, 2, 3, 4, 5, 6]
```

□□□

```
class MyList:
    def __init__(self, start):
        #self.wrapped = start[:]                    # Copy start: no side effects
        self.wrapped = []                           # Make sure it's a list here
        for x in start: self.wrapped.append(x)
    def __add__(self, other):
        return MyList(self.wrapped + other)
    def __mul__(self, time):
        return MyList(self.wrapped * time)
    def __getitem__(self, offset):
        return self.wrapped[offset]
    def __len__(self):
        return len(self.wrapped)
    def __getslice__(self, low, high):
        return MyList(self.wrapped[low:high])
    def append(self, node):
        self.wrapped.append(node)
    def __getattr__(self, name):                    # Other methods: sort/reverse/etc
        return getattr(self.wrapped, name)
    def __repr__(self):
        return repr(self.wrapped)

if __name__ == '__main__':
    x = MyList('spam')
    print(x)
    print(x[2])
    print(x[1:])
    print(x + ['eggs'])
    print(x * 3)
    x.append('a')
    x.sort()
    for c in x: print(c, end=' ')
```

```
% python mylist.py
['s', 'p', 'a', 'm']
a
['p', 'a', 'm']
['s', 'p', 'a', 'm', 'eggs']
['s', 'p', 'a', 'm', 's', 'p', 'a', 'm', 's', 'p', 'a', 'm']
a a m p s
```

□□□

□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.0□□□□□□□□□□□□□□□□□□□□□□__X__□□□□□□□□□□□□□□□□□□□

```
% python
>>> from setwrapper import Set
>>> x = Set([1, 2, 3, 4])                # Runs __init__
>>> y = Set([3, 4, 5])

>>> x & y                                # __and__, intersect, then __repr__
Set:[3, 4]
>>> x | y                                # __or__, union, then __repr__
Set:[1, 2, 3, 4, 5]

>>> z = Set("hello")                     # __init__ removes duplicates
>>> z[0], z[-1]                          # __getitem__
('h', 'o')

>>> for c in z: print(c, end=' ')        # __getitem__
...
h e l o
>>> len(z), z                            # __len__, __repr__
(4, Set:['h', 'e', 'l', 'o'])

>>> z & "mello", z | "mello"
(Set:['e', 'l', 'o'], Set:['h', 'e', 'l', 'o', 'm'])
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□multiset.py□□□□□□□□□□□□□□□□□□□

```
class MultiSet(Set):
    """
    Inherits all Set names, but extends intersect
    and union to support multiple operands; note
    that "self" is still the first argument (stored
    in the *args argument now); also note that the
    inherited & and | operators call the new methods
    here with 2 arguments, but processing more than
    2 requires a method call, not an expression:
    """

    def intersect(self, *others):
        res = []
        for x in self:                          # Scan first sequence
            for other in others:                # For all other args
                if x not in other: break        # Item in each one?
            else:                               # No: break out of loop
                res.append(x)                   # Yes: add item to end
        return Set(res)

    def union(*args):                           # self is args[0]
        res = []
        for seq in args:                        # For all args
            for x in seq:                       # For all nodes
                if not x in res:
                    res.append(x)               # Add new items to result
        return Set(res)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□intersect□□intersect□□setwrapper.Set□□multiset□□□□□□□□□□□□□□□□MultiSet□□Set□□□□□□□□□□□□□□□□

```
>>> from multiset import *
>>> x = MultiSet([1,2,3,4])
>>> y = MultiSet([3,4,5])
>>> z = MultiSet([0,1,2])

>>> x & y, x | y                                    # Two operands
(Set:[3, 4], Set:[1, 2, 3, 4, 5])

>>> x.intersect(y, z)                               # Three operands
Set:[]
>>> x.union(y, z)
Set:[1, 2, 3, 4, 5, 0]
>>> x.intersect([1,2,3], [2,3,4], [1,2,3])          # Four operands
Set:[2, 3]
>>> x.union(range(10))                              # Non-MultiSets work, too
Set:[1, 2, 3, 4, 0, 5, 6, 7, 8, 9]
```

6.□□□□□□□□□□Lister□□□□□□□□□□□□□□□□□□□□□□□□□□□□dir□□□□□□□□□□□□□□□□□□□

```
class Lunch:
    def __init__(self):                          # Make/embed Customer, Employee
        self.cust = Customer()
        self.empl = Employee()
    def order(self, foodName):                   # Start Customer order simulation
        self.cust.placeOrder(foodName, self.empl)
    def result(self):                            # Ask the Customer about its Food
        self.cust.printFood()
class Customer:
    def __init__(self):                          # Initialize my food to None
        self.food = None
    def placeOrder(self, foodName, employee):    # Place order with Employee
        self.food = employee.takeOrder(foodName)
    def printFood(self):                         # Print the name of my food
        print(self.food.name)

class Employee:
    def takeOrder(self, foodName):               # Return Food, with desired name
        return Food(foodName)

class Food:
    def __init__(self, name):                    # Store food name
        self.name = name

if __name__ == '__main__':
    x = Lunch()                                  # Self-test code
    x.order('burritos')                          # If run, not imported
    x.result()
    x.order('pizza')
    x.result()


% python lunch.py
burritos
pizza
```

8.□□□□□□□□□□□□□□□Python□□□□□□□□zoo.py□□□□□□□□□□□□□□□

```
class Animal:
    def reply(self):    self.speak()            # Back to subclass
    def speak(self):    print('spam')           # Custom message

class Mammal(Animal):
    def speak(self):    print('huh?')

class Cat(Mammal):
    def speak(self):    print('meow')

class Dog(Mammal):
    def speak(self):    print('bark')

class Primate(Mammal):
    def speak(self):    print('Hello world!')

class Hacker(Primate):  pass                    # Inherit from Primate
```

9.□□□□□□□□□□□□□□□□□□□□□□□□□parrot.py□□□□Actor□□□line□□□□□□□□□self□□□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□□self.name□self.says()□□□□□□□□□□□□□□□

```
class Actor:
    def line(self): print(self.name + ':', repr(self.says()))

class Customer(Actor):
    name = 'customer'
    def says(self): return "that's one ex-bird!"

class Clerk(Actor):
    name = 'clerk'
    def says(self): return "no it isn't..."

class Parrot(Actor):
    name = 'parrot'
    def says(self): return None

class Scene:
    def __init__(self):
        self.clerk = Clerk()                    # Embed some instances
        self.customer = Customer()              # Scene is a composite
        self.subject  = Parrot()
```

```
def action(self):
    self.customer.line()                    # Delegate to embedded
    self.clerk.line()
    self.subject.line()
```

□□□□ □□□□□

□□□35□“□□□□ □□□”□□□□

1.try/except□□□□oops□□□□□□□oops.py□□□□□□□

2.□□□

```
import sys, traceback

def safe(entry, *args):
    try:
        entry(*args)                        # Catch everything else
    except:
        traceback.print_exc()
        print('Got', sys.exc_info()[0], sys.exc_info()[1])

import oops
safe(oops.oops)
```

```
% python safe2.py
Traceback (innermost last):
  File "safe2.py", line 5, in safe
    entry(*args)                            # Catch everything else
  File "oops.py", line 4, in oops
    raise MyError, 'world'
hello: world
Got hello world
```

4.□□□

```
# Find the largest Python source file in a single directory

import os, glob
dirname = r'C:\Python30\Lib'

allsizes = []
allpy = glob.glob(dirname + os.sep + '*.py')
for filename in allpy:
    filesize = os.path.getsize(filename)
    allsizes.append((filesize, filename))

allsizes.sort()
print(allsizes[:2])
print(allsizes[-2:])

# Find the largest Python source file in an entire directory tree

import sys, os, pprint
if sys.platform[:3] == 'win':
    dirname = r'C:\Python30\Lib'
```

```
else:
    dirname = '/usr/lib/python'

allsizes = []
for (thisDir, subsHere, filesHere) in os.walk(dirname):
    for filename in filesHere:
        if filename.endswith('.py'):
            fullname = os.path.join(thisDir, filename)
            fullsize = os.path.getsize(fullname)
            allsizes.append((fullsize, fullname))

allsizes.sort()
pprint.pprint(allsizes[:2])
pprint.pprint(allsizes[-2:])
```

Find the largest Python source file on the module import search path

```
import sys, os, pprint
visited = {}
allsizes = []
for srcdir in sys.path:
    for (thisDir, subsHere, filesHere) in os.walk(srcdir):
        thisDir = os.path.normpath(thisDir)
        if thisDir.upper() in visited:
            continue
        else:
            visited[thisDir.upper()] = True
        for filename in filesHere:
            if filename.endswith('.py'):
                pypath = os.path.join(thisDir, filename)
                try:
                    pysize = os.path.getsize(pypath)
                except:
                    print('skipping', pypath)
                allsizes.append((pysize, pypath))

allsizes.sort()
pprint.pprint(allsizes[:3])
pprint.pprint(allsizes[-3:])
```

Sum columns in a text file separated by commas

```
filename = 'data.txt'
sums = {}

for line in open(filename):
    cols = line.split(',')
    nums = [int(col) for col in cols]
    for (ix, num) in enumerate(nums):
        sums[ix] = sums.get(ix, 0) + num

for key in sorted(sums):
    print(key, '=', sums[key])
```

Similar to prior, but using lists instead of dictionaries for sums

```
import sys
```

```
filename = sys.argv[1]
numcols = int(sys.argv[2])
totals = [0] * numcols

for line in open(filename):
    cols = line.split(',')
    nums = [int(x) for x in cols]
    totals = [(x + y) for (x, y) in zip(totals, nums)]

print(totals)
```

Test for regressions in the output of a set of scripts

```
import os
testscripts = [dict(script='test1.py', args=''),           # Or glob script/args dir
               dict(script='test2.py', args='spam')]

for testcase in testscripts:
    commandline = '%(script)s %(args)s' % testcase
    output = os.popen(commandline).read()
    result = testcase['script'] + '.result'
    if not os.path.exists(result):
        open(result, 'w').write(output)
        print('Created:', result)
    else:
        priorresult = open(result).read()
        if output != priorresult:
            print('FAILED:', testcase['script'])
            print(output)
        else:
            print('Passed:', testcase['script'])
```

Build GUI with tkinter (Tkinter in 2.6) with buttons that change color and grow

```
from tkinter import *                       # Use Tkinter in 2.6
import random
fontsize = 25
colors = ['red', 'green', 'blue', 'yellow', 'orange', 'white', 'cyan', 'purple']

def reply(text):
    print(text)
    popup = Toplevel()
    color = random.choice(colors)
    Label(popup, text='Popup', bg='black', fg=color).pack()
    L.config(fg=color)

def timer():
    L.config(fg=random.choice(colors))
    win.after(250, timer)

def grow():
    global fontsize
    fontsize += 5
    L.config(font=('arial', fontsize, 'italic'))
    win.after(100, grow)

win = Tk()
```

```
L = Label(win, text='Spam',
          font=('arial', fontsize, 'italic'), fg='yellow', bg='navy',
          relief=RAISED)
L.pack(side=TOP, expand=YES, fill=BOTH)
Button(win, text='press', command=(lambda: reply('red'))).pack(side=BOTTOM,fill=X)
Button(win, text='timer', command=timer).pack(side=BOTTOM, fill=X)
Button(win, text='grow', command=grow).pack(side=BOTTOM, fill=X)
win.mainloop()
```

Similar to prior, but use classes so each window has own state information

```
from tkinter import *
import random
class MyGui:
    """
    A GUI with buttons that change color and make the label grow
    """
    colors = ['blue', 'green', 'orange', 'red', 'brown', 'yellow']

    def __init__(self, parent, title='popup'):
        parent.title(title)
        self.growing = False
        self.fontsize = 10
        self.lab = Label(parent, text='Gui1', fg='white', bg='navy')
        self.lab.pack(expand=YES, fill=BOTH)
        Button(parent, text='Spam', command=self.reply).pack(side=LEFT)
        Button(parent, text='Grow', command=self.grow).pack(side=LEFT)
        Button(parent, text='Stop', command=self.stop).pack(side=LEFT)

    def reply(self):
        "change the button's color at random on Spam presses"
        self.fontsize += 5
        color = random.choice(self.colors)
        self.lab.config(bg=color,
                font=('courier', self.fontsize, 'bold italic'))

    def grow(self):
        "start making the label grow on Grow presses"
        self.growing = True
        self.grower()

    def grower(self):
        if self.growing:
            self.fontsize += 5
            self.lab.config(font=('courier', self.fontsize, 'bold'))
            self.lab.after(500, self.grower)

    def stop(self):
        "stop the button growing on Stop presses"
        self.growing = False

class MySubGui(MyGui):
    colors = ['black', 'purple']          # Customize to change color choices

MyGui(Tk(), 'main')
MyGui(Toplevel())
MySubGui(Toplevel())
```

```
mainloop()

# Email inbox scanning and maintenance utility

"""
scan pop email box, fetching just headers, allowing
deletions without downloading the complete message
"""

import poplib, getpass, sys
mailserver = 'your pop email server name here'              # pop.rmi.net
mailuser = 'your pop email user name here'                  # brian
mailpasswd = getpass.getpass('Password for %s?' % mailserver)

print('Connecting...')
server = poplib.POP3(mailserver)
server.user(mailuser)
server.pass_(mailpasswd)

try:
    print(server.getwelcome())
    msgCount, mboxSize = server.stat()
    print('There are', msgCount, 'mail messages, size ', mboxSize)
    msginfo = server.list()
    print(msginfo)
   for i in range(msgCount):
       msgnum = i+1
       msgsize = msginfo[1][i].split()[1]
       resp, hdrlines, octets = server.top(msgnum, 0)       # Get hdrs only
       print('-'*80)
       print('[%d: octets=%d, size=%s]' % (msgnum, octets, msgsize))
       for line in hdrlines: print(line)

       if input('Print?') in ['y', 'Y']:
           for line in server.retr(msgnum)[1]: print(line) # Get whole msg
       if input('Delete?') in ['y', 'Y']:
           print('deleting')
           server.dele(msgnum)                              # Delete on srvr
       else:
           print('skipping')
finally:
   server.quit()                                        # Make sure we unlock mbox
input('Bye.')                                           # Keep window up on Windows

# CGI server-side script to interact with a web browser

#!/usr/bin/python
import cgi
form = cgi.FieldStorage()                               # Parse form data
print("Content-type: text/html\n")                      # hdr plus blank line
print("<HTML>")
print("<title>Reply Page</title>")                      # HTML reply page
print("<BODY>")
if not 'user' in form:
    print("<h1>Who are you?</h1>")
else:
    print("<h1>Hello <i>%s</i>!</h1>" % cgi.escape(form['user'].value))
```

```
print("</BODY></HTML>")
```

Database script to populate and query a MySql database

```
from MySQLdb import Connect
conn = Connect(host='localhost', user='root', passwd='darling')
curs = conn.cursor()
try:
    curs.execute('drop database testpeopledb')
except:
    pass                                            # Did not exist

curs.execute('create database testpeopledb')
curs.execute('use testpeopledb')
curs.execute('create table people (name char(30), job char(10), pay int(4))')

curs.execute('insert people values (%s, %s, %s)', ('Bob', 'dev', 50000))
curs.execute('insert people values (%s, %s, %s)', ('Sue', 'dev', 60000))
curs.execute('insert people values (%s, %s, %s)', ('Ann', 'mgr', 40000))

curs.execute('select * from people')
for row in curs.fetchall():
    print(row)

curs.execute('select * from people where name = %s', ('Bob',))
print(curs.description)
colnames = [desc[0] for desc in curs.description]
while True:
    print('-' * 30)
    row = curs.fetchone()
    if not row: break
    for (name, value) in zip(colnames, row):
        print('%s => %s' % (name, value))

conn.commit()                                       # Save inserted records
```

Database script to populate a shelve with Python objects

```
# see also Chapter 27 shelve and Chapter 30 pickle examples

rec1 = {'name': {'first': 'Bob', 'last': 'Smith'},
        'job': ['dev', 'mgr'],
        'age': 40.5}

rec2 = {'name': {'first': 'Sue', 'last': 'Jones'},
        'job': ['mgr'],
        'age': 35.0}

import shelve
db = shelve.open('dbfile')
db['bob'] = rec1
db['sue'] = rec2
db.close()
```

Database script to print and update shelve created in prior script

```
import shelve
db = shelve.open('dbfile')
for key in db:
    print(key, '=>', db[key])

bob = db['bob']
bob['age'] += 1
db['bob'] = bob
db.close()
```

□□□□

□□□□Python□□□□□□□□□Mark Lutz□Python□□□□□□□□□□□□□□Python□□□□□□□

Mark□O'Reilly□□□□Programming Python□□□Python Pocket Reference□□□□□□□□□□2009□□□□□□□□3□□Mark□1992□□□□□Python□1995□□□□□□□Python□□□□□1997□□□□□Python□□□□□2009□□□□□□□□225□Python□□□□□□□□□□□□□3500□□□□□□□□□□□25□□□□Python□□□□□□□□□□□□□□□□□□□□

□□□Mark□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□25□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□wood rat□□□□□□□□□□□□